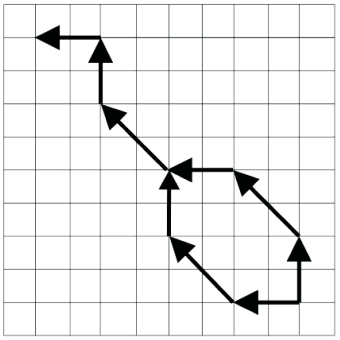
Pătcaș Csaba György



Informatikafeladatok megoldása haladó módszerekkel

Feladatgyűjtemény

Presa Universitară Clujeană / Kolozsvári Egyetemi Kiadó

Pătcaş Csaba György

Informatikafeladatok megoldása haladó módszerekkel

Feladatgyűjtemény

Presa Universitară Clujeană / Kolozsvári Egyetemi Kiadó

2019

A kiadvány megjelenését az NTP-HTTSZ-M-19-0005 kódszámú pályázaton keresztül az Emberi Erőforrások Minisztériuma támogatta. A pályázatot az Emberi Erőforrások Minisztériuma megbízásából az Emberi Erőforrás Támogatáskezelő hirdette meg.

Szaklektorok:
Dr. Bodó Zalán egyetemi docens
Dr. Gaskó Noémi egyetemi docens

ISBN 978-606-37-0687-5

Borítóterv: Bodó Zalán

Universitatea Babeş-Bolyai
Presa Universitară Clujeană
Director: Codruţa Săcelean
Str. Hasdeu nr. 51
400371 Cluj-Napoca, România
Tel./fax: (+40)-264-597.401
E-mail: editura@editura.ubbcluj.ro
http://www.editura.ubbcluj.ro/

Tartalomjegyzék

1. fejezet

Előszó

Könyvünkről Ezt a feladatgyűjteményt elsősorban segédanyagnak szánjuk az *Informatikafeladatok megoldása haladó módszerekkel* nevű tantárgyhoz, mely opcionálisan választható a Babeş–Bolyai Tudományegyetem Matematika–Informatika Karán elsőéves egyetemisták számára. A tantárgy fő célja a hallgatók megismertetése az algoritmikai jellegű egyéni és csapatos programozói versenyekkel, valamint olyan haladóbb szintű algoritmusok bemutatása, amelyekre más tantárgyak keretein belül nem kerül sor.

Könyvünk hasznos lehet olyanok számára is, akik érdeklődnek az informatikaversenyek és az algoritmusok világa iránt és mélyíteni szeretnék tudásukat ilyen téren.

A bemutatott feladatokat a szerző javasolta az elmúlt 15 év során, különböző megyei, megyeközi, országos és nemzetközi szintű versenyeken. Néhány feladat kijelentésében kisebb módosításokat végeztünk el a versenyen javasolt változathoz képest, hogy általánosabb, nehezebb

feladatokat kapjunk. A feladatok egy része közös munka eredménye dr. Ionescu Klárával, akinek ezúton is köszönetet mondok az évek során nyújtott mindennemű segítségért.

A fejezetek felépítése A második fejezet megoldási módszer szerint csoportosítva tartalmaz feladatokat, azok számára, akik egy-egy módszer alkalmazását szeretnék elmélyíteni. A gráfelméleti, dinamikus programozással megoldható és különféle adatszerkezetek ismeretét igénylő feladatokat nehézség szerinti növekvő sorrendbe rendeztük alfejezetenként.

A megoldási módszer ismerete túl nagy segítségnek bizonyulhat rutinosabb feladatmegoldók számára. Számukra javasoljuk a harmadik fejezetben megtalálható három-három feladatból álló feladatsorokat, melyeket igyekeztünk minél színesebbre összeállítani, mind nehézségi szint, mind megoldási módszer szempontjából. Ezeket a feladatsorokat versenyszimulációkra is fel lehet használni, a megoldó ismereteinek függvényében 2-4 óra munkaidőt javasolunk egy-egy feladatsor kidolgozására.

A negyedik fejezetben mindegyik javasolt feladathoz található útmutatás, magyarázat formájában. Itt sok esetben alternatív megoldásokról is említést teszünk és elemezzük az algoritmusok hatékonyságát.

Végül az ötödik fejezetben a feladatok megoldásai találhatóak meg, melyeket C/C++ nyelvben implementáltunk. Mivel versenykörnyezetben legtöbbször néhány óra leforgása alatt kell a feladatokat megoldani és egyes versenyeken az implementálás ideje is rangsorolási szempont, a versenyzők ritkán tartják be a klasszikus softwarefejlesztési környe-

zetben használt programtervezési elveket. Megoldásainkban mi is ezt a filozófiát követjük és átlépjük az objektumorientált programozás, valamint a modularitás elveit, sok helyen rugalmasan használunk globális változókat, változóneveink gyakran párbetűsek, a megjegyzések szinte teljes mértékben hiányoznak. Úgy gondoljuk, hogy mindezek ellenére a programkódunk könnyen értelmezhető a feladat és az útmutatás ismeretében, illetve hasznos forrás lehet a versenyeken használt stílus és különböző trükkök elsajátítására. A kódban többnyire angol változó- és függvényneveket használunk és a ritka megjegyzéseink is angol nyelvűek, mivel a programozásban ez univerzális gyakorlatnak számít és vegyes nyelvi háttérrel rendelkező csapatok esetén sokat könnyít egymás kódjának megértésében.

Javaslatok a könyv használatához A legfontosabb javaslatunk, hogy ne lépjünk túl hamar a feladatról az útmutatásra, vagy az útmutatásról a megoldásra.

A feladat elolvasása után töltsünk minimum egy óra időt intenzív koncentrálással, amikor külső zavaró tényezők kizárásával, csak a feladat megoldásán gondolkozzunk. Ha úgy gondoljuk, hogy találtunk egy megoldást, két dologról kell megbizonyosodnunk: a helyességről és a hatékonyságról. Próbáljuk ki az algoritmusunkat papíron vagy fejben, különböző esetekre. Az, hogy a feladatban szereplő példára helyesen működik, még nem garancia semmire! Próbáljunk matematikai szempontból helyes bizonyítást találni az algoritmusunk helyességére, vagy legalább intuitíven bizonyítani azt. Létezhet ellenpélda a megközelítésünkre? Miért nem?

Ha jelentős gondolkodási idő után sem jutottunk semmire, akkor térjünk csak át az útmutatás elolvasására. Próbáljuk megérteni minden részletét a leírásnak és ezután implementálni az algoritmust! Jelentős különbség van az ötlet megértése és annak programformába öntése között, úgyhogy ne ugorjnk át az implementálási szakaszt! Szánjunk legalább két óra koncentrált munkaidőt az implementálásra, csak ezután nézzük meg a megoldást!

Ha sikerült implementálni a megoldást, ellenőrizzük, hogy mennyi időbe telt a programot megírni. Az esetek nagy százalékában egy gyakorlott feladatmegoldónak a legkomplexebb feladatok megoldása is legtöbb fél órába telik, a megoldási ötlet kidolgozása után. Ha nekünk több időbe tellett, hasonlítsuk össze kódunkat a megoldásban szereplővel. Lehetett volna valamit egyszerűbben megvalósítani? Miután levontuk a következtetéseket, pihentessük a feladatot pár napot és próbáljuk ismét megoldani. Remélhetőleg másodjára már sokkal gyorsabb lesz az implementálás és letisztultabb a kód.

2. fejezet

Feladatok megoldási módszer szerint

2.1. Gráfelméleti feladatok

2.1.1. Project management

Megyei olimpia, 11-12. osztály, 2009

Egy software gyártó cégnél nagy projekten dolgoznak. A projekt n ($n \in \mathbb{N}$) fejlesztési fázis végrehajtásából áll, melyeket az $1, 2, \ldots, n$ számokkal jelölünk. Bizonyos fázisokat végre lehet hajtani párhuzamosan (egyidőben) is, viszont más fázisok végrehajtását nem lehet elkezdeni addig, amíg adott fázisok végrehajtása be nem fejeződött.

Követelmény

Írjatok programot, amely meghatározza:

1. azt a minimális t időt, ami alatt a projektet be lehet fejezni

2. minden k ($k \in \{1, 2, \ldots, n\}$) fázisra azt a c_k legkorábbi időpontot, amikor a k. fázis legkorábban elkezdődhet és azt a legkésőbbi d_k időpontot, amikor a k. fázis legkésőbben elkezdődhet anélkül, hogy befolyásolná a projekt teljes végrehajtási idejét.

Bemeneti adatok

A `pm.in` bemeneti állomány tartalma:

- az első sor tartalmazza az n természetes számot, amely a projekt fázisainak számát jelöli
- a második sorban n természetes szám található egy-egy szóközzel elválasztva, melyek minden fázis végrehajtásához szükséges időt jelölik
- a következő n sor mindegyike tartalmaz egy m_k természetes számot és egy a sorozatot, mely m_k természetes számot tartalmaz egy-egy szóközzel elválasztva, jelöljük ezeket $a_1, a_2, \ldots, a_{m_k}$-val. m_k azoknak a fázisoknak a számát jelöli, melyeket a k. fázis elkezdése előtt be kell fejezni, míg az a sorozat elemei azon fázisok sorszámait, amelyeket be kell fejezni a k. fázis elkezdése előtt.

Kimeneti adatok

A `pm.out` kimeneti állomány $n + 1$ sort kell tartalmazzon. Az első sorba írjuk a t természetes számot és a következő k sor mindegyikébe egy-egy szóközzel elválasztva a c_k és d_k számokat.

Megszorítások és pontosítások

- $0 \leq n \leq 100$, $n \in \mathbb{N}$
- Egy fázis befejezéséhez szükséges idő nem fogja meghaladni az 1.000.000-t.
- A projekt végrehajtása a 0. időpillanatban kezdődik.
- Nem lesznek körkörös függőségek (a projekt minden esetben befejezhető).
- Az 1-es követelmény megoldása 40%-ot ér, míg a 2-es követelmény megoldása 30-30%-ot.

Példa

pm.in	pm.out
7	11
2 3 5 3 3 3 2	0 3
0	0 0
0	3 3
1 2	2 5
1 1	2 5
1 1	8 8
3 3 4 5	8 9
1 3	

2.1.2. Cannibal

ECN Sapientia, 2007

A folyó egyik partján n kannibál és m misszionárius várakozik, hogy átkelhessenek a túlsó partra egy csónak segítségével, amely legtöbb k személy szállítására alkalmas. A probléma csak annyi, hogy ha a folyó valamelyik oldalán, vagy a csónakban, a kannibálok száma meghaladja a misszionáriusok számát, akkor a kannibálok megeszik a misszionáriusokat, ami természetesen nem történhet meg.

Követelmény

Döntsük el, hogy lehetséges-e a kannibálok és misszionáriusok számára a túloldalra való átkelés. Ha igen, határozzuk meg, hogy minimálisan hányszor kell a csónak átkeljen a folyón (bármelyik irányba) a feladat megoldásához.

Bemeneti adatok

A bemeneti állomány számos esetet tartalmaz. Mindegyik sorban megtalálhatóak az n, m és k értékek szóközökkel elválasztva. Az utolsó sor esetében $n = m = k = -1$ és ezt a sort nem kell feldolgozni.

Kimeneti adatok

A bemeneti állományban található minden esetre (kivéve az utolsót) írjunk a kimeneti állomány mindegyik sorába egy-egy számot. Ha a feladat nem megoldható, ez a szám legyen -1, egyébként legyen a minimális átkelések száma.

Megszorítások és pontosítások

- $0 \leq n \leq m \leq 500$
- $1 \leq k \leq 1000$

- A bemenet legtöbb 70.000 esetet tartalmazhat.
- Az esetek legalább 99,9%-ában $m \leq 50$

Példa

cannibal.in	cannibal.out
1 1 1 1 1 2 -1 -1 -1	-1 1

2.1.3. Domino2

Grigore Moisil megyeközi, 11-12. osztály, 2006

Adott n dominó. Határozzuk meg, hogy lehetséges-e egy olyan dominósorozat felépítése, amely betartja a dominó játék szabályát és tartalmazza az összes dominót. A dominó játék szabálya azt mondja ki, hogy két egymás után következő dominó egymással szomszédos oldalán ugyanannak a számnak kell szerepelnie. A dominókat bármilyen sorrendben kiválaszthatjuk és el is forgathatjuk.

Bemeneti adatok

A `domino2.in` állomány első sora a dominók n számát tartalmazza. A következő n sor mindegyikében két természetes szám található egy szóközzel elválasztva, melyek az adott dominó két oldalára írt számot jelképezik.

Kimeneti adatok

A `domino2.out` állomány első sorába az 1-es számot kell írni, ha létezik megoldás és nullát, ha nincs. Ha létezik megoldás, a következő

n sor írja le a felépített sorozatot abban a sorrendben, ahogyan abban a dominók szerepelnek. Mindegyik sor az adott dominó sorszámát tartalmazza, majd egy szóközt és 0-t, ha a dominót nem forgattuk meg, vagy 1-t, ha a dominót megforgattuk.

Megszorítások és pontosítások

- $1 \leq n \leq 50.000$
- A dominókra írt számok 0 és 9 közöttiek.

Példa

domino2.in	domino2.out
12	1
1 4	3 0
3 7	11 0
1 5	12 1
2 4	9 1
2 5	8 1
2 6	2 0
2 7	7 1
3 4	6 0
4 6	10 1
5 6	5 1
5 7	4 0
6 7	1 1

2.1.4. Online

Grigore Moisil megyeközi, 11-12. osztály, 2008

Egy képzeletbeli országban úthálózatot kell építeni n város között. A közlekedésügyi miniszter m céghez fordult, melyeket megkért, hogy becsüljék meg két város közötti direkt út megépítésének árát. Az m ajánlatból, amelyek m különböző út megépítésére vonatkoztak, a miniszter kiválasztott egy minimális számú város párt amelyek közé direkt út épüljön, úgy, hogy bármelyik városból bármelyikbe el lehessen jutni direkt vagy indirekt módon és az utak megépítésének költsége minimális legyen.

Az építkezés k hét múlva kellett volna kezdődjön, viszont ebben a periódusban a közlekedésügyi miniszter hetente kapott egy-egy új ajánlatot, amelyek vagy egy újabb város pár összekötésére vonatkoztak, vagy olyan városok összekötésére, amelyekre már érkezett ajánlat. A miniszternek aktualizálnia kell a projektet és meg kell határoznia hetente, melyik utakat építse meg.

Követelmény

Határozzátok meg az eredeti projektet, majd a k új ajánlat mindegyikére egy-egy minimális költségű új projektet.

Bemeneti adatok

Az `online.in` állomány első sorában az n és m természetes számok találhatóak egy szóközzel elválasztva, melyek a városok és a kezdeti ajánlatok számát jelölik. A következő m sor mindegyikében három szám szerepel egy-egy szóközzel elválasztva: az első kettő azon városok sorszámát jelöli, amelyekre az ajánlat vonatkozik, a harmadik meg a két várost összekötő út árát. A következő sorban a k természetes szám

található, amely azon hetek számát jelöli, melyek során a projektet javítani kell. A következő k sor mindegyikében három szám szerepel egy-egy szóközzel elválasztva, ugyanazzal a jelentéssel, mint fentebb.

Kimeneti adatok

Az `online.out` állományba a projektek költségét kell írni: a kezdeti projektét, majd a k aktualizálás utáni költségeket. Egy projekt költségén a kiválasztott utak árainak összegét értjük.

Megszorítások és pontosítások

- $1 \leq n \leq 200$
- $1 \leq m \leq 10.000$
- $1 \leq k \leq 10.000$
- $1 \leq$ két város közötti út ára ≤ 250
- A városok közötti utak kétirányúak.

Példa

online.in	online.out
5 7	10
1 2 1	8
2 3 7	8
1 5 5	
3 4 3	
4 5 4	

1 3 2 2 4 6 2 3 5 2 1 4 3	

2.2. Dinamikus programozás módszerével megoldható feladatok

2.2.1. Persely

Grigore Moisil megyeközi, 7-8. osztály, 2008

Pistike több érmét gyűjtött össze egy malacperselybe. Mielőtt elkezdte a pénzgyűjtést, megmérte az üres persely súlyát, így most pontosan tudja a perselyben található érmék összsúlyát. Pistike meg szeretne vásárolni egy könyvet, ezért meg szeretné határozni a perselyben található pénzösszeget anélkül, hogy széttörje azt. Ahhoz, hogy megtudhassa a perselyben található összeget, megkéri Annát, hogy mérjen le egy-egy érmét a perselyben található típusokból.

Követelmény

Határozzátok meg azt az összeget, amely biztosan megtalálható malacperselyben!

Bemeneti adatok

A `p.in` állomány első sorában az S természetes szám található, amely az érmék összsúlyát jelöli, valamint egy n természetes szám, ami az érmék típusainak számát jelöli. A következő n sor mindegyikében egy-egy érmetípus leírása található: az első szám a súlyát és a második szám az értékét jelöli.

Kimeneti adatok

A `p.out` állományba azt az összeget írjuk, amely biztosan megtalálható a perselyben!

Megszorítások és pontosítások

- $1 \leq n \leq 100$
- $1 \leq S \leq 10.000$
- $1 \leq$ *egy érme súlya* ≤ 100
- $1 \leq$ *egy érme értéke* ≤ 100

Példa

p.in	p.out
15 4 2 1 3 2 6 5 4 10	8

2.2.2. Domino

Grigore Moisil megyeközi, 10. osztály, 2005

Adott n dominó, amelyekből sorozatok építhetőek a következő szabályok betartásával:

- Az első dominódarab kötelezően a sorozat része.
- A következő darabokat a megadott sorrendben vesszük figyelembe és mindegyikről eldöntjük, hogy beletesszük-e a sorozatba, vagy sem.
- Mindegyik darabot a meglévő sorozat valamelyik végéhez illeszthetjük abban a pozicióban amelyben meg volt adva, vagy 180 fokkal elfordítva. Ha úgy döntünk, hogy nem tesszük bele a sorozatba, akkor félre rakjuk és többet nem használhatjuk.
- Egy dominót akkor lehet a sorozat valamelyik végére tenni, ha a sorozat végén lévő dominón szereplő szám (az, amelyik nincs a már megépített sorozathoz ragasztva) megegyezik az új dominó azon számával, amelyet a sorozathoz ragasztunk.

Követelmény

Építsük meg a leghosszabb dominósorozatot!

Bemeneti adatok

A `domino.in` állomány első sorában a dominók n száma található. A következő n sor mindegyikében két x és y szám található, amelyek az adott dominón szereplő számokat jelölik.

Kimeneti adatok

A `domino.out` állományba egyetlen számot kell írni, a leghosszabb dominósorozat hosszát, amelyet a fentebbi szabályok betartásával meg lehet építeni.

Megszorítások és pontosítások

- $1 \leq n \leq 100.000$
- $1 \leq x, y \leq 9$
- A dominódarabokat meg lehet forgatni mielőtt a sorozat valamelyik végére helyeznénk őket.

Példa

domino.in	domino.out
6 1 2 1 6 2 3 1 4 2 3 4 3	5

2.2.3. Tango

Országos olimpia, 9. osztály, 2010

D és S első táncversenyükre készülnek és éppen a tangó koreográfián dolgoznak. Annak ellenére, hogy ez lesz az első versenyük, ők már

ismernek n táncfigurát és kiszámolták, hogy melyik hány ütemet tart. Emellett ismerik a tangózene jellegzetességeit is, amely nyolc ütemből álló zenei frázisokból áll. Ahhoz, hogy a koreográfia szép legyen, minden frázis kezdeténél egy új figurának kell kezdődnie. Mivel D és S nagyon szeretnek együtt táncolni, elő szeretnének készíteni egy tangó koreográfiát egy olyan zeneszámra, amely pontosan k ütemből áll.

Két koreográfiát akkor tekintünk különbözőnek, ha az eltáncolt figurák sorozata különbözik.

Követelmény

Határozzátok meg azon koreográfiák lehetséges számának osztási maradékát 999.983-al, amelyek betartják a fenti szabályokat és kizárólag a D és S által ismert figurákból állnak (már nincs elég idő a versenyig új figurák megtanulására).

Bemeneti adatok

A `tango.in` állomány első sora az n és k számokat tartalmazza egy szóközzel elválasztva. A második sorban pontosan n természetes szám található egy-egy szóközzel elválasztva, melyek a figurák hosszait jelölik.

Kimeneti adatok

A `tango.out` állományba a lehetséges koreográfiák számának 999.983-al való osztási maradékát kell írni.

Megszorítások és pontosítások

- $1 \leq n \leq 100.000$
- $1 \leq k \leq 2.000.000.000$

- $1 \leq$ egy figura hossza ≤ 50
- A tesztek 30%-ában egy adott hosszúságú figurából csak egy darab lesz.
- A tesztek 50%-ában $n \leq 30$
- A tesztek 70%-ában a figurák hossza kizárólag a $\{2, 4, 6, 8\}$ halmazból kerül ki.

Példa

tango.in	tango.out
3 16 1 1 8	66049

Magyarázat: A zeneszám 16 ütemből, vagyis 2 frázisból áll. Jelöljük a figurákat A-val, B-vel és C-vel, abban a sorrendben, ahogyan a bemeneti állományban szerepelnek, tehát A hossza 1, B hossza 1 és C hossza 8. Az első frázis felépíthető az A és B figurák bármilyen sorozatából, tehát összesen $2^8 = 256$-féleképpen. Egy másik lehetőség az első frázis esetén egyetlen C figura eltáncolása, tehát összesen 257 lehetőségünk van az első frázis eltáncolására. Ugyanennyi lehetőség van a második frázis esetén is, tehát összesen $257 \cdot 257 = 66049$ lehetséges koreográfiánk van.

2.2.4. Dday

Gazeta de Informatică, 2006

Mint minden évben, Hollandiában idén is készülnek az úgynevezett „Dominónap”-ra. Ennek során megpróbálják megdönteni a legtöbb ledöntött dominó világrekordját, vagyis megpróbálnak egy olyan dominósorozatot felépíteni, amely magától felborul miután az első dominót kézzel felborítják. Ahhoz, hogy az esemény minél látványosabb legyen, a dominókat úgy helyezik el, hogy miután felborultak, bizonyos képeket alakítsanak ki. Ezeknek a képeknek minden évben van egy közös tematikájuk.

Ebben az évben a szervező a ti segítségeteket kéri a költségek optimalizálához. Tudjuk, hogy m egységnyi pénz áll rendelkezésére a dominók megvásárlására és k önkéntes azok elhelyezésére. Természetesen maximalizálni szeretné a megrendelt dominók számát a rendelkezésére álló erőforrásokból.

A dominókat különböző cégektől lehet megrendelni, amelyek már gyárilag kiszínezik őket megfelelően, úgy, hogy miután felborultak, az a kép alakuljon ki belőlük, amelyet a cég javasol az ezévi tematikához. Minden ilyen képre ismerjük azon dominók számát amelyekből a kép áll, ezek árát és a képhez tartozó dominók felrakásához szükséges önkéntesek számát.

Bemeneti adatok

A `dday.in` állomány első sorában három természetes szám található egy-egy szóközzel elválasztva: n, m és k. A következő n sor mindegyikében három természetes szám található: a képhez szükséges

dominók száma, a kép ára és a kép felrakásához szükséges önkéntesek száma.

Kimeneti adatok

A `dday.out` kimeneti állomány első sorába négy számot kell írni: a kiválasztott képek számát, az összes dominó számát, a képek összárát és a szükséges önkéntesek számát. A második sorban a kiválasztott ajánlatok sorszámai kell szerepeljenek egy-egy szóközzel elválasztva, növekvő sorrendbe rendezve.

Megszorítások és pontosítások

- $1 \leq n \leq 100$
- $1 \leq m \leq 100$
- $1 \leq k \leq 100$
- Egy képhez tartozó dominók száma nem haladja meg az 1.000.000-t.
- Egy kép felrakásához szükséges önkéntesek száma nem haladja meg a 100-at.
- Egy kép ára nem haladja meg a 100 pénzegységet.
- Egy önkéntes csak egyetlen kép kirakásában segíthet.

Példa

dday.in	dday.out
4 10 20 20000 1 1 55000 6 11 50000 4 9 80000 7 12	2 105000 10 20 2 3

2.3. Adatszerkezetekkel kapcsolatos feladatok

2.3.1. Raktár

Nemes Tihamér első forduló, 3. kategória, 2011

Egy raktárba különböző termékek érkeznek, melyeknek van árukódjuk és áruk.

Követelmény

Írjatok programot, amely a következő műveleteket valósítja meg:

- `TERMEK(nev, ar)`: új termék érkezik a raktárba *nev* árukóddal és *ar* árral. Garantált, hogy ezelőtt nem létezett a raktárban termék azonos kóddal.
- `AR(nev)`: visszatéríti a *nev* árukóddal rendelkező termék árát. Ha nem létezik termék ezzel a kóddal, 0-t térít vissza.

Megszorítások és pontosítások

- A `TERMEK` műveletet legtöbb 4000-szer hívjuk meg.
- Az `AR` műveletet legtöbb 50.000-szer hívjuk meg.
- Az árukód egy pontosan négy karakterből álló karaktersorozat, mely csak az angol ábécé kisbetűit tartalmazhatja.
- Az ár egy 1 és 100 közötti egész szám.

Példa

raktar.in	raktar.out
AR(abcd)	0
TERMEK(abcd,1)	1
TERMEK(xyzq,3)	0
AR(abcd)	3
AR(ffff)	3
TERMEK(qwer,3)	
AR(xyzq)	
AR(qwer)	

2.3.2. Ékszerek

Nemes Tihamér első forduló, 3. kategória, 2009

Egy előkelő ékszerüzletbe csak gazdag vevők járnak. Mivel mindegyik vevő szeret dicsekedni a vagyonával, mindegyik az üzletben lévő legdrágább ékszert veszi meg.

Követelmény

Írjatok programot az üzlet, mint adatszerkezet kezelésére, és a következő feladatok elvégzésére:

- `EKSZER(x)`: Egy x értékű ékszer érkezik az üzletbe.
- `VEVO`: Egy vevő érkezik az üzletbe, visszatéríti az üzletben lévő legdrágább ékszer értékét és eltávolítja azt az üzletből.

Megszorítások és pontosítások

- $1 \leq x \leq 1000$ és x egész szám
- Az `EKSZER` műveletet legtöbb 10.000-szer fogjuk meghívni.
- A `VEVO` műveletet nem hívjuk meg, ha az üzletben nincs ékszer.

Példa

ekszer.in	ekszer.out
EKSZER(1)	1
VEVO	5
EKSZER(1)	3
EKSZER(5)	2
EKSZER(3)	
VEVO	
VEVO	
EKSZER(2)	
VEVO	

2.3.3. Cég

Nemes Tihamér első forduló, 3. kategória, 2010

Egy cégnek n kirendeltsége van, melyeket a cégvezető 1-től n-ig számozott meg. A cégek időről időre jelentéseket küldenek a cégvezetőnek a bevétel változását illetően, amelyek alapján a cégvezetőnek bizonyos statisztikákra van szüksége.

Követelmény

Írjatok programot, amely a következő műveleteket valósítja meg:

- `JELENTES(i, dif)`: az i. kirendeltség jelenti, hogy dif különbség van az aktuális és az ezelőtti bevétel között.
- `STATISZTIKA(i1, i2)`: visszatéríti az *i1.* és *i2.* kirendeltség közötti kirendeltségek aktuális bevételeinek összegét, az *i1.* és *i2.* kirendeltséget is beleszámítva.

Megszorítások és pontosítások

- $1 \leq n \leq 4000$
- $1 \leq i \leq n$
- $-1000 \leq dif \leq 1000$
- $1 \leq i1 \leq i2 \leq n$
- A `JELENTES` műveletet legtöbb 10.000-szer fogjuk meghívni.
- A `STATISZTIKA` műveletet legtöbb 40.000-szer fogjuk meghívni.

- Feltételezzük, hogy kezdetben minden kirendeltség bevétele nulla.

Példa

ceg.in	ceg.out
10	0
STATISZTIKA(1,10)	-3
JELENTES(1,-3)	2
JELENTES(3,5)	20
STATISZTIKA(1,2)	
STATISZTIKA(1,6)	
JELENTES(3,-2)	
JELENTES(7,20)	
STATISZTIKA(1,10)	

2.3.4. Motel

Grigore Moisil megyeközi, 10. osztály, 2009

Az Encián motelben évekkel előre is fogadnak foglalásokat az oda-érkező n turistacsoport számára. Ismervén a napot, amellyel kezdve foglalni lehet és amelyet 1-el számozunk, a csoportok megjelölik az első és az utolsó napot, amelyet a motelben szeretnének tölteni. Az Encián tulajdonosa mindegyik csoport számára szeretne egy tradicionális előadást szervezni, amire csak a motel dísztermében kerülhet sor, ahova egyszerre csak egy turistacsoport fér be. Ezeken az előadásokon fel fog lépni egy előadó, aki megszabja azt az n napot, amikor el tud jönni az előadásokra.

Követelmény

Segítsetek a tulajdonosnak eldönteni, hogy melyik turistacsoportnak melyik napra szervezze meg az előadást.

Bemeneti adatok

A `motel.in` állomány első sorában egy n természetes szám található, amely a turistacsoportok számát, illetve azon napok számát jelöli, amikor az előadó el tud jönni előadást tartani. A következő n sor mindegyikében két természetes szám található egy szóközzel elválasztva, melyek a turistacsoportok által igényelt első és utolsó napot jelölik. A következő n sor mindegyikében egy természetes szám található, amely azokat a napokat jelöli, amikor az előadó igénybe vehető.

Kimeneti adatok

A `motel.out` kimeneti állomány n sort kell tartalmazzon. Mindegyik sorba két természetes számot kell írni egy szóközzel elválasztva. Az első szám a turistacsoport sorszáma a második meg az előadás napjának sorszáma, abban a sorrendben, ahogyan a bemeneti állományban szerepelnek. Abban az esetben, ha létezik legalább egy turistacsoport, amelyik számára nem lehet meghatározni, hogy melyik napon legyen a neki szánt előadás, a kimeneti állományba kizárólag két 0-t kell írni egy szóközzel elválasztva.

Megszorítások és pontosítások

- $1 \leq n \leq 4000$
- $1 \leq$ egy nap sorszáma ≤ 30.000

Példák

motel.in	motel.out
5 12 23 100 120 5 15 45 60 35 56 5 48 50 110 13	3 1 1 5 5 2 4 3 2 4

Megjegyzés: Az `5 2` és `4 3` párosítások helyett egy másik helyes válasz lett volna az `5 3` és `4 2` is.

motel.in	motel.out
3 5 20 135 156 30 50 8 48 7	0 0

3. fejezet

Feladatsorok

3.1. Első feladatsor

3.1.1. Tekaj

Gazeta de Informatică, 2005

A Marson a legelterjedtebb sportág nem a futball, hanem egy *Tekaj* nevű játék. A labdarúgáshoz hasonlóan egy bajnokságban két szezon van (oda- és visszavágók), viszont egy mérkőzésnek lehet nulla, egy, vagy két győztese is. Akárcsak a fociban, mindkét szezonban mindegyik csapat játszik mindegyik csapat ellen, tehát ha n csapatunk van, akkor mindegyik $n - 1$ meccset fog játszani az első szezonban és $n - 1$-et a másodikban.

A bajnokság befejeztével a bajnokot egy csoportos mérkőzésen döntik el, melyet két csoport játszik egymás ellen, mindkét csoportot a bajnokságban szereplő csapatok egy része alkotja. A csoportok ter-

mészetesen diszjunktak kell legyenek és a következő feltételek alapján határozzák meg őket: az első csoportban lévő csapatok által megnyert mérkőzések összege osztható kell legyen a második csoportban lévő csapatok által megnyert mérkőzések összegével. Ha többféleképpen ki lehet választani a csoportokat, szervezési okokból azt a változatot fogják választani, amelyben a legkevesebb csapat szerepel.

Bemeneti adatok

A `tekaj.in` állomány első sorában a csapatok n száma szerepel. A második sorban pontosan n szám található egy-egy szóközzel elválasztva, mely a csapatok által megnyert meccsek számát jelöli.

Kimeneti adatok

A `tekaj.out` kimeneti állománynak négy sora kell legyen. Az első sorba az első csoportban szereplő csapatok számát írjuk, a második sorba meg a csapatok sorszámait. A harmadik sorban a második csoportban lévő csapatok száma szerepel, míg a negyedik sorban ezen csapatok sorszámai.

Megszorítások és pontosítások

- $0 \leq n \leq 100.000$
- Egy csapat által megnyert mérkőzések száma 0 és $2 \cdot (n-1)$ között lesz.
- Ha több minimális csapatot tartalmazó megoldás létezik, bármelyiket meg lehet adni.

Példa

tekaj.in	tekaj.out
5 2 3 5 6 7	1 4 1 1

Magyarázat: Egy lehetséges megoldás lehetett volna az {1, 2} és a {3} (mindkét csoport összesen 5 mérkőzést nyert meg), viszont ez nem teljesítette volna a minimalitás követelményét. Egy másik helyes megoldás a {4} és a {2} lett volna.

3.1.2. Bishop

ECN Sapientia, 2007

Adott egy sakktábla n sorral és m oszloppal. Számítsuk ki, hányféleképpen helyezhetünk el rá k futót, úgy, hogy azok ne üssék egymást.

Bemeneti adatok

A `bishop.in` állomány mindegyik sora egy-egy esetet tartalmaz, amelyeket három, szóközökkel elválasztott, n, m és k számmal írunk le. Az utolsó sor esetében $n = m = k = -1$ és ezt nem kell feldolgozni.

Kimeneti adatok

Mindegyik esetre a `bishop.out` állomány egy-egy sorába írjuk a lehetőségek számát.

Megszorítások és pontosítások

- $1 \leq n, m \leq 13$
- $1 \leq k \leq n \cdot m$
- Két futó akkor üti egymást, ha ugyanazon az átlón találhatóak.

Példa

bishop.in	bishop.out
2 2 2	4
3 3 1	9
8 8 64	0
-1 -1 -1	

3.1.3. Zip

Grigore Moisil megyeközi, 11-12. osztály, 2008

Egy tetszőleges egyetemista egy tetszőleges egyetemen azt a házi feladatot kapta egy tetszőleges tantárgyból, hogy tömörítőprogramot írjon. Az egyetemista a következő algoritmusra gondolt:

- A tömörítendő állományt felosztjuk m darab k karaktert tartalmazó részre.
- Két egymás után következő darabra meghatározzuk a leghosszabb olyan szakaszt az első darab végéről, amelyik megjelenik a második darab elején is.

- A tömörítés úgy történik, hogy a fent meghatározott közös részeket csak egyszer írjuk ki.

Egy programhiba miatt az eredeti állomány darabjai közé más k hosszúságú darabok is odakeveredtek és a darabok sorrendje is megváltozott. Ugyancsak a hiba következtében, ha eredetileg léteztek egyforma darabok, előfordulhat, hogy ezek kevesebbszer szerepelnek.

Követelmény

Tömörítsétek az eredeti állományt, tudván, hogy ez azokból a darabokból áll, amelyek az optimális tömörítéshez vezetnek, vagyis a tömörített állomány hossza minimális.

Bemeneti adatok

A `zip.in` állomány első sorában három szám szerepel egy-egy szóközzel elválasztva: n (az összes darab száma), m (az eredeti állományban található darabok száma) és k (egy darab mérete). A következő n sor mindegyikében k karakter szerepel, melyek a darabok tartalmát jelölik.

Kimeneti adatok

A `zip.out` állományba egyetlen számot kell írni, a sűrített állomány méretét.

Megszorítások és pontosítások

- $1 \leq m \leq n \leq 500$
- $1 \leq k \leq 500$

Példa

zip.in	zip.out
5 2 4 rado dora arad oboa aaaa	5

Magyarázat: Az eredeti állomány az `arad` és `rado` darabokból állt. A leghosszabb szakasz, amelyik teljesíti a követelményeket a `rad`, tehát a tömörített állomány tartalma `arado` lesz, melynek mérete öt karakter. Egy másik lehetőség a tömörített állomány tartalmára az `aaaaa` karaktersor.

3.2. Második feladatsor

3.2.1. Összeg

Grigore Moisil megyeközi, 9. osztály, 2008

Egy kockás lapon körberajzoltunk egy téglalapot. A téglalapban szereplő minden kockában egy természetes szám szerepel, melyeknek pontosan a fele egyenlő 0-val.

Be szeretnénk járni egy útvonalat a téglalapon belül, mely érinti az összes nem-nulla elemet úgy, hogy az útvonalon mindegyik mezőt egyszer érintjük. A számokat úgy járjuk be, hogy egyik számról a

vele szomszédos számokra léphetünk, vagyis arra a nyolc szomszédra, amely azonos soron, oszlopon, vagy átlón található.

Az *útvonal összegét* úgy definiáljuk, hogy azon $k \cdot a_{ij}$ szorzatok összege, melyekre k a mező sorszáma, abban a sorrendben, ahogy az útvonalban szerepel és a_{ij} a mezőn található szám.

$$\textit{Útvonal összege} = \sum_{k=1}^{[n \cdot m/2]} k \cdot a_{ij}$$

Követelmény

Határozzátok meg a legkisebb összegű útvonalat, valamint ennek összegét!

Bemeneti adatok

Az `osszeg.in` állomány első sorában két természetes szám található, n és m, melyek a téglalap méreteit jelölik, n a sorok számát és m az oszlopokét. A következő n sor mindegyikében m szám található egy-egy szóközzel elválasztva.

Kimeneti adatok

Az `osszeg.out` állomány első sorába a legkisebb összegű útvonal összegét kell írni. A második sor fogja leírni az útvonalat, vagyis az összes nullától különböző számot a téglalapból, abban a sorrendben, ahogy bejártuk őket. A számokat egy-egy szóközzel válasszuk el!

Megszorítások és pontosítások

- $0 \leq n, m \leq 8$
- Az $n \cdot m$ szorzat mindig páros lesz.
- $0 \leq$ a téglalapban szereplő számok ≤ 100
- Mindig létezni fog megoldás.

Példa

osszeg.in	osszeg.out
3 4 1 0 10 0 4 3 0 8 0 0 2 0	1 70 10 8 2 3 4 1

Magyarázat: $1 \cdot 10 + 2 \cdot 8 + 3 \cdot 2 + 4 \cdot 3 + 5 \cdot 4 + 6 \cdot 1 = 70$

3.2.2. Törpék

Grigore Moisil megyeközi, 9. osztály, 2008

Aprajafalván n ház található 1-től n-ig számozva. Törpapa megtudta a Jótündértől, hogy a falut veszély fenyegeti. Egy varázsital segítségével Törpapa láthatatlanná tudja tenni a faluban található házakat, amíg a veszély elmúlik. Sajnos a varázsitalhoz szükséges egyik összetevőből már csak nagyon kevés van és nincs már elegendő idő elmenni az erdőbe és pótolni azt. Emiatt Törpapa csak azokat a házakat tudja láthatatlanná tenni, amelyek három (nem feltétlenül különböző) egyenesen találhatóak.

Követelmény

Határozzátok meg, hogy a falut el lehet-e tüntetni teljes egészében. Ha igen, adjátok meg az egyeneseket, amelyek mentén a törpék háza található, vagyis azokat az egyeneseket, amelyek „lefedik” a törpék házait.

Bemeneti adatok

A `torpek.in` állomány első sora a házak n számát tartalmazza. A következő n sor mindegyike két egész számot tartalmaz egy szóközzel elválasztva, melyek a házak koordinátáit jelölik.

Kimeneti adatok

Ha a törpék házait nem lehet lefedni legtöbb három egyenessel, akkor a `torpek.out` kimeneti állományba azt kell írni, hogy „A torpek veszelyben vannak”. Ellenkező esetben a kimeneti állomány első sorába azt kell írni, hogy „A torpek meg fognak menekulni”. Ebben az esetben a következő három sor mindegyikébe két-két természetes számot kell írni egy szóközzel elválasztva, amelyek azon házak sorszámait jelölik, melyek egy-egy egyenest határoznak meg.

Megszorítások és pontosítások

- $2 \leq n \leq 5000$
- $-30.000 \leq$ a házak koordinátái ≤ 30.000
- Ha több megoldás létezik, a kimeneti állományba bármelyiket ki lehet írni.
- Két ház nem fog egyazon ponton elhelyezkedni.

Példa

torpek.in	torpek.out
7	A torpek meg fognak menekulni
0 0	1 2
0 1	1 4
0 2	1 7
1 1	
2 2	
2 0	
1 0	

Magyarázat: Az egyenes, amely összeköti az 1-es házat a 2-es házzal, lefedi a 3-as házat is. Az egyenes, amely összeköti az 1-es házat a 4-es házzal, lefedi az 5-ös házat is. Az egyenes, amely összeköti az 1-es házat a 7-es házzal, lefedi a 6-os házat is.

3.2.3. Kovács János

ECN Sapientia, 2008

A tegnapi buli után Kovács János háza egy szemétdomb. A karórája a hűtőben, pár narancs az ágyban és cigaretták a mikrohullámú sütőben. János minél hamarabb rendet szeretne rakni, mivel az édesanyja hamarosan meglátogatja és ilyen állapotban mégsem láthatja a házat.

János házának n szobája van és alaprajza leírható egy irányítatlan gráffal, melynek n csúcsa és m éle van. János a hálószobában van (melynek sorszáma 1) és a takarítást az előszobában szeretné

befejezni (melynek sorszáma n). Összesen k tárgyat kell visszarakni az eredeti helyére. Mindegyik tárgyra ismerjük, hogy éppen melyik szobában található és melyik szobában van a helye. János legtöbb p tárgyat tud magával vinni egyszerre. Ő a következő műveleteket tudja végrehajtani:

- Átmegy egy szomszédos szobába.
- Felvesz egy tárgyat, amely abban a szobában található, mint ő.
- Letesz egy tárgyat az aktuális szobába, ha az megegyezik a tárgy célszobájával.

Ezen műveletek mindegyike pontosan tizenhárom másodpercet vesz igénybe.

Bemeneti adatok

A `kovacsjanos.in` állomány első sora a tesztesetek l számát fogja tartalmazni. Mindegyik esetet a következőféleképpen írjuk le. Az első sor tartalmazza az n és m számokat egy szóközzel elválasztva. A következő m sor mindegyikében két szomszédos szoba sorszáma található egy szóközzel elválasztva. A következő sor tartalmazza a k és p számokat egy szóközzel elválasztva. A következő k sor mindegyike egy tárgy kiindulási és végső szobájának sorszámát tárolja egy szóközzel elválasztva. A bemeneti állományban üres sorok is előfordulhatnak tetszőleges helyen.

Kimeneti adatok

A `kovacsjanos.out` állományba írjátok a bulin résztvevő személyek számát, János életkorát és édesanyjának lánykori nevét. Tulajdonképpen ezt hagyjuk inkább és írjátok csak ki, hogy hány másodperc

alatt lehet kitakarítani a házat. Mindegyik tesztesetre adott választ külön sorba írjátok.

Megszorítások és pontosítások

- $1 \leq l \leq 15$
- $1 \leq n \leq 50$
- $0 \leq m \leq 1000$
- $0 \leq k \leq 5$
- $1 \leq p \leq 5$
- A gráf összefüggő lesz.

Példa

kovacsjanos.in	kovacsjanos.out
1 4 3 1 2 2 3 4 3 1 2 2 3	65

Magyarázat: János a következő műveleteket végzi el: átmegy az 1-es szobából a 2-es szobába, felveszi az ott található tárgyat, átmegy a 2-es szobából a 3-as szobába, leteszi a tárgyat, majd átmegy a 3-as szobából a 4-es szobába.

4. fejezet

Útmutatások

4.1. Gráfelméleti feladatok

4.1.1. Project management

A megoldáshoz szükséges algoritmus a szakirodalomban *Critical Path Method*-ként ismert [21].

Először felépítjük a függőségek gráfját és hozzá adunk két fiktív csúcsot, melyek a kezdeti, illetve a végső pillanatot jelölik. Ezeket számozhatjuk 0-val és $n + 1$-el.

A második lépés a gráf szintekre bontása oly módon, hogy minden ív egy alacsonyabb szintről egy magasabb szintre mutasson. Ez a lépés több módszerrel kivitelezhető, például a klasszikus topologikus rendezés algoritmusával. Ha ezt a lépést kihagyjuk és a csúcsokat 1, 2, . . . , n sorrendben járjuk be, a pontszám 50%-át érhetjük el.

Az utolsó lépés a kért értékek kiszámítása. Jelöljük $earliest_k$-val a minimális időpontot, amikor a k. fázis elkezdődhet és $latest_k$-val a

maximális időpontot, amikor a k. fázis elkezdődhet, anélkül, hogy a projekt teljes időtartamát megváltoztatná. A rekurzív összefüggések könnyen levezethetők:

$earliest_0 = 0$

$earliest_k = max(earliest_j + time_j)$, ahol a k függ a j-től

$latest_{n+1} = earliest_{n+1}$

$latest_k = min(latest_j - time_k)$, ahol a j függ a k-tól

Az *earliest* értékeket a szintek növekvő sorrendjében, a *latest* értékeket meg a szintek csökkenő sorrendjében számítjuk ki. Szomszédsági mátrixként tárolva a gráfot, a végső bonyolultság $O(n^2)$. Ha szomszédsági listákat használunk, akkor a transzponált gráfot is tárolnunk kell, ekkor a bonyolultságot lecsökkenthetjük $O(m)$-re. Mindkét módszerrel elérhető a maximális pontszám.

4.1.2. Cannibal

Építsünk fel egy gráfot, melynek minden csúcsát egy (a, b, c) számhármassal címkézünk, ahol $a = \overline{0, n}$, $b = \overline{0, m}$ és $c = \overline{0, 1}$. Egy csúcsnak a következő lesz a jelentése: a kannibál van a folyó egyik oldalán, b misszionárius van a folyó ugyanazon oldalán és a csónak a c-vel jelölt oldalon található. Világos, hogy nem szükséges a túloldalon található kannibálok illetve misszionáriusok számát tárolni, mivel a számuk külön-külön konstans (n illetve m), ezért az egyik oldalon lévők száma egyértelműen meghatározza a másik oldalon lévők számát.

Kössük össze az (a_1, b_1, c_1) és (a_2, b_2, c_2) csúcsokat egy éllel, ha eljuthatunk (a_1, b_1, c_1)-ból (a_2, b_2, c_2)-ba egyetlen átkeléssel. Ezután az átalakítás után egyértelművé válik, hogy a feladat a minimális

hosszúságú út meghatározását kéri az $(n, m, 0)$ és $(0, 0, 1)$ csúcsok között. Mivel a gráf súlyozatlan, a leghatékonyabb módszer a szélességi bejárás [28, 27].

A feladat egyetlen nehézségét az esetek nagy száma jelentette. Ez viszont könnyen áthidalható azon esetek előre kiszámításával, melyekre $n, m \leq 50$ (ennek pár perc alatt le kell futnia). Ezek után a szélességi bejárást csak a maradék 0,01% nagy eset megoldására alkalmazzuk.

4.1.3. Domino2

Építsünk fel egy irányítatlan gráfot, melynek csúcsait számozzuk 0-tól 9-ig. Minden dominóra adjunk hozzá egy élet a dominó két oldalán szereplő számnak megfelelő csúcsok közé. Ekkor a feladat egy olyan lánc megkeresését kéri ebben a gráfban, mely minden élen egyszer halad át, ezt *Euler láncnak* nevezzük.

Egy hatékony és könnyen implementálható módszer ennek a megkeresésére *Hierholzer algoritmusa* [6]. A bemeneti adatok mérete megengedi ennek rekurzív implementálását és a gráf szomszédsági mátrixszal való tárolását. A keresést egy páratlan fokszámú csúcsból indítjuk, ha van ilyen, ha nincs, akkor egy olyan csúcsból, melynek nem nulla a fokszáma.

A megoldás visszakereséséhez használhatunk 100 darab vermet, amelyeket egy 10 soros és 10 oszlopos mátrixban tárolunk és mindegyik veremben megjegyezzük azon dominók sorszámait, amelyeknek az adott számok vannak a két oldalán.

Megjegyezzük, hogy a megfelelő futási sebesség eléréséhez szükséges lehet a kiíratáskor `endl` helyett `'\n'` használatával új sort kezdeni,

ugyanis az előbbi mindig üríti a puffert és lényegesen lassúbb.

4.1.4. Online

A feladat a minimális költségű feszítőfa megkeresését kéri abban az esetben, amikor a gráf változik minden lépésben. Az első ötlet a klasszikus algoritmusok végrehajtása $k+1$-szer, így a használt algoritmustól és annak implementálási módjától függően különböző bonyolultságú megoldásokat kapunk. *Kruskal algoritmusát* [26] használva a következő bonyolultságokat kaphatjuk: $O(k{\cdot}n{\cdot}m)$[1], $O(k{\cdot}(m \log m{+}n^2))$[2] vagy $O(k \cdot m \log m)$[3]. *Prim algoritmusát* [24, 30, 10] használva $O(k \cdot n^2)$[4] vagy $O(k \cdot m \log n)$[5] bonyolultságot kapunk.

Azonban a fenti megközelítések egyike sem fut le időben a megadott megszorításokra. A maximális pontszám eléréséhez két lehetőséget javasolunk.

$O(m \log m + k \cdot \alpha(n))$ bonyolultságot kaphatunk a következő algoritmussal. Megkeressük a kezdeti minimális feszítőfát Kruskal algoritmusával, diszjunkthalmaz-erdőkkel implementálva [5, 19, 33]. A fának $n-1$ éle lesz, ehhez egyenként hozzávesszük a megadott k él mindegyikét, úgy, hogy ár szerint beszúrjuk a megfelelő helyre a meglévő $n-1$ él közé. Az így kapott n élből álló rendezett sorozaton végigmegyünk Kruskal algoritmusának stratégiája szerint és megkeres-

[1] naívan implementálva, gráfbejárással ellenőrizve, ha két csúcs azonos komponenshez tartozik

[2] hovatartozási vektorral

[3] diszjunkthalmaz-erdőkkel

[4] naívan implementálva

[5] min-kupacokkal implementálva

sük az új minimális feszítőfát. Mindig lesz egy él az n-ből, amely nem tartozik a fához, ezt töröljük, majd az eddigieket ismételjük mindegyik élre a megadott k-ból.

A másik javasolt megoldás bonyolultsága $O(n^2 + k \cdot n)$. Először megkeressük a minimális feszítőfát Prim algoritmusával. A következő k lépés mindegyikében két eset lehetséges:

1. A megadott él része az aktuális minimális feszítőfának. Ha a megadott él költsége kisebb, mint a fában szereplő élé, felülírjuk a fában szereplő élet és frissítjük a fa költségét. Ha az él költsége nagyobb vagy egyenlő, mint a fában szereplő élé, a fa változatlanul marad.

2. A megadott él nem része a fának. Mivel minden fa maximális számú élet tartalmaz a körmentesség szempontjából, az új él bevezetésével egy kört alakítunk ki. Ahhoz, hogy ismét egy fához jussunk, ebből a körből ki kell törölni egy élet. Ahhoz, hogy az így kapott fa költsége minimális legyen, a körből a maximális költségű élet fogjuk kitörölni.

Megfelelően implementálva a fenti két esetet lekezelhetjük egyben. Ahhoz, hogy egy él törlését konstans időben elvégezhessük, a fát szomszédsági listákkal tároljuk és egy külön mátrix (i, j) elemében tárolunk egy mutatót az i. csúcs szomszédsági listájának azon elemére, amely a j csúccsal való összeköttetést biztosítja.

Minden új élre egy mélységi bejárással megkeressük az utat az él két végpontja között (mivel fáról van szó, pontosan egy ilyen út létezik), majd ellenőrizzük, hogy az úton található legnagyobb költség

nagyobb-e, mint az él költsége. Az első esetben az út egyetlen élt fog tartalmazni, a másodikban legalább kettőt.

Az eredeti minimális feszítőfa meghatározásakor oda kell rá figyelnünk, hogy a bemenet egy multigráfot tartalmaz és két csúcs között több él is meg lehet adva. Ezek közül csak a legkisebb költségűt kell figyelembe vennünk.

4.2. Dinamikus programozás módszerével megoldható feladatok

4.2.1. Persely

Egy rövid elemzés után levonhatjuk a következtetést, hogy a feladat tulajdonképpen a minimális összeget kéri, melyet olyan érmék alkothatnak, amelyeknek az összsúlya S.

A feladatot megpróbálhatjuk megoldani különböző mohó típusú stratégiákkal, az érméket rendezve bizonyos kritériumok szerint és ebben a sorrendben kiválasztva mindig, amennyit lehet. Ezek a megoldások a pontszám 30-40%-át érik el. Ha több ilyen megoldást összekombinálunk és kiválasztjuk az eredmények közül a legjobbat, akkor elérhetünk 60%-ig.

A maximális pontszám eléréséhez szükség van a dinamikus programozás módszerének ismeretére [2, 3]. Felépítünk egy vektort, melyet 0-tól S-ig indexelünk és az i. pozición azt a legkisebb értéket tároljuk, amelyet olyan érmék alkothatnak, melyek összsúlya i. Kezdőértéknek a vektor mindegyik eleme egy nagy értéket kap, kivéve a 0. pozí-

ción lévő elemet, mely 0 lesz. A sorozatot a következőféleképpen számíthatjuk ki:

```
for i = 0, s
  for j = 1, n
    if (i > suly[j] && vekt[i] > vekt[i - suly[j]] +
        ertek[j])
      vekt[i] = vekt[i - suly[j]] + ertek[j]
```

4.2.2. Domino

A bemeneti adatok méretéből látszik, hogy egy négyzetes időbonyolultságú algoritmus túl lassú lenne, ezért egy lineáris algoritmust fogunk használni.

A dinamikus programozás módszerét alkalmazzuk [2, 3]. Egy mátrixot használunk, melynek az i, j indexén lévő eleme a leghosszabb olyan dominósorozat hosszát jelöli, melynek egyik végén az i, másik végén a j szám található. Tehát a mátrix sorait és oszlopait 0-tól 9-ig indexeljük.

Mivel az első dominó kötelezően a sorozat része kell legyen, a megfelelő elemét 1-es értékkel inicializáljuk, más dominóval nem próbálunk új sorozatot kezdeni. Ahhoz, hogy elkerüljük, hogy ugyanazt a dominót többször ragasszuk önmagához, a fenti mátrixnak két másolatát használjuk, az egyik jelöli az aktuális változatot, amelyből előállítjuk az újat, majd ez a kettő szerepet cserél.

4.2.3. Tango

A megoldás két részből áll. Az első részben ki kell számolnunk, hogy egy frázis hányféleképpen táncolható el. Erre több módszer is lehetséges, megoldható akár egy megfelelően optimizált visszalépéses kereséssel (backtracking), akár a hátizsák feladathoz (knapsack) hasonló dinamikus programozással [29]. Természetesen a 8-nál hosszabb figurákat figyelmen kívül hagyhatjuk, hiszen ezek nem lehetnek a megoldás részei.

A második részben a kapott eredményt a $k/8$-adik hatványra kell emelni a gyorshatványozás algoritmusával. Végül ezt megszorozzuk az esetlegesen a zeneszám végén megmaradt részleges frázis felépítésének módozataival. Megoldásunk végső bonyolultsága $O(\log k)$.

4.2.4. Dday

A feladat tulajdonképpen egy kétdimenziós hátizsák probléma különösebb bonyodalmak nélkül [29]. A $c_{i,j,p}$ háromdimenziós tömb mindegyik elemébe kiszámoljuk a legtöbb dominó számát, amelyet el lehet érni az első i ajánlat, j pénzegység és p önkéntes felhasználásával. Ugyan a c_i értékek kiszámításához csak a c_{i-1} értékeket használjuk, de a megoldás visszakeresése úgy a legegyszerűbb, ha a teljes tömböt megtartjuk a memóriában, amit a feladat korlátai kényelmesen megengednek. A visszakereséshez egy másik tömbbe lementjük a legnagyobb ajánlat sorszámát, amellyel az adott értéket kaptuk az első tömbben. A feladat megoldásához meg kell keresnünk a legnagyobb értéket az összes $c_{n,j,p}, j = \overline{1,m}, p = \overline{1,k}$ közül.

4.3. Adatszerkezetekkel kapcsolatos feladatok

4.3.1. Raktár

Több megoldás létezik, ezeket soroljuk fel a következőkben, a kevésbé hatékonyaktól a leghatékonyabbig. Az alábbiakban jelöljük n-el a `TERMEK` alprogram hívásainak számát ($n \leq 4000$), m-el az `AR` alprogram hívásainak számát ($m \leq 50.000$) és l-el az árukód maximális hosszát ($l = 4$).

1. Egy tömb végére teszi az új adatot, lineáris bejárással keres:

   - `TERMEK`: $O(1)$ időbonyolultság
   - `AR`: $O(n \cdot l)$ időbonyolultság

   A teljes műveletsor időbonyolultsága $O(n + m \cdot n \cdot l)$, vagyis kb. $8 \cdot 10^8$ művelet a legrosszabb esetben.

2. Egy rendezett tömbbe szúrja be az új adatot, binárisan keres [23]:

   - `TERMEK`: $O(n \cdot l)$ időbonyolultság
   - `AR`: $O(\log n \cdot l)$ időbonyolultság

   A teljes műveletsor időbonyolultsága $O(n^2 \cdot l + m \cdot \log n \cdot l)$, vagyis kb. $6{,}5 \cdot 10^7$ művelet a legrosszabb esetben.

3. Egy kiegyensúlyozott bináris fát használ [1, 17]:

- `TERMEK`: $O(\log n \cdot l)$ időbonyolultság
- `AR`: $O(\log n \cdot l)$ időbonyolultság

A teljes műveletsor időbonyolultsága $O(n \cdot \log n \cdot l + m \cdot \log n \cdot l)$, vagyis kb. $2{,}5 \cdot 10^6$ művelet a legrosszabb esetben.

4. Prefix-fát (trie) [9] vagy hasító-táblát (hash-table) használ [7, 8]:

   - `TERMEK`: $O(l)$ időbonyolultság
   - `AR`: $O(l)$ időbonyolultság

   A teljes műveletsor időbonyolultsága $O(n \cdot l + m \cdot l)$, vagyis kb. $2 \cdot 10^5$ művelet a legrosszabb esetben.

4.3.2. Ékszerek

Több megoldás létezik, ezeket soroljuk fel a következőkben a kevésbé hatékonyaktól a leghatékonyabbig.

1. Egy 10.000 elemű tömbben tárolja az ékszerek értékeit és amikor vevő érkezik, megkeresi a maximumot, majd törli.

   Ekkor az `EKSZER` művelet bonyolultsága O(1), vagyis konstans időben fut. A `VEVO` művelet bonyolultsága $O(n)$, a legrosszabb esetben kb. 20.000 alapművelet elvégzése szükséges: megkeresi a maximumot (10.000) és "felé tolja" a tőle jobbra található elemeket (10.000). A láncolt listákkal, rendezett tömbökkel, vagy rendezett láncolt listákkal történő megvalósítás is ebbe a kategóriába tartozik.

2. Egy 1000 elemű tömbben tárolja, hogy melyik értékű ékszerből hány darab van az üzletben.

   Ekkor az `EKSZER` művelet bonyolultsága O(1), vagyis konstans időben fut. A `VEVO` művelet a legrosszabb esetben kb. 1000 alapműveletet végez: 1000-től indulva lefele "lépeget" a maximum (az első nullától különböző elem) megtalálásáig. Ha egy globális változóban tárolja a maximális értéket, elkerüli az 1000-től való lépegetést, de ha nincs több maximum értékű ékszer (vagyis a csökkentés után a tömb megfelelő eleme nullázódik), aktualizálnia kell a maximumot lefele lépegetéssel.

3. Kupac (max-heap) struktúrát használ [35].

   Ekkor mindkét művelet bonyolultsága $O(\log n)$.

4.3.3. Cég

Több megoldás létezik, ezeket soroljuk fel a következőkben, a kevésbé hatékonyaktól a leghatékonyabbig.

1. Egy tömbben tárolja mindegyik kirendeltség bevételét.

   Ekkor a `JELENTES` művelet $O(1)$, a `STATISZTIKA` művelet $O(n)$ időbonyolultságú, az összműveletek száma $1{,}6 \cdot 10^8$.

2. Részösszegeket tárol egy tömbben, melynek i. eleme az első i kirendeltség bevételének összegét tárolja.

   Ekkor a `JELENTES` művelet $O(n)$, a `STATISZTIKA` művelet $O(1)$ időbonyolultságú, az összműveletek száma $4 \cdot 10^7$.

3. Minden kirendeltség bevétele mellett tárolja $\sqrt{n}$ hosszúságú darabonként is a részösszegeket.

   Ekkor mindkét művelet időbonyolultsága $O(\sqrt{n})$, az összműveletek száma $3{,}2 \cdot 10^6$.

4. Intervallumfát [4] vagy binárisan indexelt fát [32, 14] használ a részösszegek tárolására.

   Ekkor mindkét művelet időbonyolultsága $O(\log n)$, az összműveletek száma $6 \cdot 10^5$.

4.3.4. Motel

Ha egyszerű rendezésen alapuló greedy algoritmussal próbáljuk a feladatot megoldani, nem fogunk sikerrel járni. Például, ha a turisták által kért intervallumokat végpont szerinti növekvő sorrendbe rendezzük, akárcsak az előadó által megadott napokat és így próbáljuk párosítani őket, hibás választ kapunk az alábbi példára:

```
2
1 4
2 3
1
2
```

A végpont szerinti rendezés után a `[2,3]` intervallum lesz az első és ehhez próbálnánk az `1`-es napot párosítani, de nem lehet, mivel nem tartozik az intervallumhoz. Így arra a következtetésre jutnánk,

hogy nincs megoldás, miközben párosíthatjuk az `[1,4]`-et az `1`-hez és a `[2,3]`-at a `2`-höz.

Ugyancsak helytelen lenne az a megoldás, amelyben az intervallumokat kezdőpontjuk szerint rendeznénk. Tekintsük ekkor az alábbi bemenetet:

```
2
1 4
2 3
2
4
```

Az `[1,4]` intervallumhoz rendelnénk a `2`-es pontot és a `4`-esnek nem jutna pár, így megint azt a választ adnánk, hogy nincs megoldás, ami helytelen.

A gráfelméleti algoritmusokban jártasak számára kézenfekvő megoldás lehet egy páros gráf felépítése és ebben egy maximális párosítás keresése. Ha ennek a párosításnak a kardinalitása n, akkor van megoldásunk, ha kisebb, akkor nincs. Ez a megoldás implementálható $O(n \cdot m)$ [16, 11, 13] vagy $O(\sqrt{n} \cdot m)$ [18, 20] bonyolultságban. A legrosszabb esetben m nagyságrendileg n^2-hez közelít, viszont a gyakorlatban erre az esetre nagyon könnyű megoldást találni, hiszen bármilyen párosítás optimális megoldáshoz vezet.

Egy ennél egyszerűbb megoldás a következő: az előadó által megadott napokat növekvő sorrendben bejárva, mindegyikhez azt az intervallumot rendeljük, amelyiknek a végpontja a legkisebb és tartalmazza az adott napot. Ezt a megoldást naívan implementálva $O(n^2)$

bonyolultságú algoritmust kapunk, amely a gyakorlatban az előző megoldáshoz hasonló futási idővel fog rendelkezni.

A maximális pontszám eléréséhez a fenti ötletet hatékonyabban kell implementálnunk. Ehhez a min-kupac (min-heap) [35] nevű adatstruktúrát használjuk. A kupac rendezési kritériuma az intervallumok végpontja lesz.

Először rendezzük az intervallumokat a kezdőpontjuk szerint és a napokat is növekvő sorrendbe. Az i. lépésben megkeressük azt az intervallumot, amelyiket az i. naphoz párosítsunk. Hozzáadjuk a kupachoz azokat az eddig nem hozzáadott intervallumokat, amelyek tartalmazzák az i. napot, majd kitöröljük a kupacból azokat, amelyek nem tartalmazzák. Ha a kupac üressé válik, akkor nincs megoldásunk, ellenkező esetben az i. napot párosítjuk a kupac gyökerében lévő elemmel. Végső bonyolultság: $O(n \log n)$.

4.4. Első feladatsor

4.4.1. Tekaj

A feladat két minimális diszjunkt részhalmaz meghatározását kéri, melyekre igaz, hogy az első részhalmazban lévő elemek összege osztható a második részhalmazban lévő elemek összegével. Az eredeti n elemű sorozat 0 és $2 \cdot n - 2$ közötti elemeket tartalmazott.

Az első észrevétel, amely szükséges a feladat megoldásához, hogy mindig létezni fog két, egy-egy elemet tartalmazó részhalmaz, ami teljesíti a feladat követelményeit. Ez bizonyítható a *Dirichlet-féle skatulya-elv* segítségével [12].

Mielőtt bemutatnánk a megoldás megkeresésének módját, megjegyezzük, hogy ha létezik egy 0 értékű elem, a feladat triviálissá válik, hiszen minden szám osztja a nullát. Ebben az esetben az egyetlen dolog, amire figyelni kell az, hogy a másik kiválasztott szám ne legyen nulla, mivel a $\frac{0}{0}$ tört matematikai szempontból nem értelmezhető.

Ha a sorozat nem tartalmaz nullás értéket, akkor az elemek az $[1, 2 \cdot n - 2]$ intervallumhoz tartoznak. A megoldáshoz észrevesszük, hogy bármely természetes szám felírható $a \cdot 2^b$ alakban, ahol a egy páratlan szám. Csoportosíthatjuk az n számot úgy, hogy mindegyik csoportban azok a számok szerepeljenek, melyekre az a értéke azonos. Viszont az $[1, 2 \cdot n - 2]$ intervallumban pontosan $n - 1$ páratlan szám található, tehát a skatulya-elv alapján létezni fog legalább egy csoport, amelyik több mint egy elemet tartalmaz. Mivel egy szám mindig osztja a vele egy csoportban lévő nagyobb számokat, a megoldás felépítése innentől egyszerűvé válik.

4.4.2. Bishop

Az első ötlet az, hogy generáljuk a k futó összes elhelyezési módját az $n \cdot m$ mezőre, úgy, hogy ne üssék egymást. Ezen javíthatunk, ha észrevesszük, hogy a táblának összesen $n + m - 1$ átlója van és természetesen egy átlóra csak egyetlen futót helyezhetünk el, mivel másképpen az egy átlón lévő futók ütnék egymást. Felhasználva ezt az észrevételt, kiválaszthatunk k átlót az $n+m-1$-ből minden lehetséges módon, majd generálhatjuk ezekre a futók elhelyezését, mindegyik kiválasztott átlóra egy futót.

Egy kis sakktudással javíthatunk a fenti megoldáson. A sakktábla

mezői fehér vagy fekete színűek a koordinátáik összegének paritása függvényében. Egy fehér mezőn található futó soha nem üthet egy fekete mezőn található futót, ami érvényes fordítva is. Ennek a tulajdonságnak a felhasználásával jutunk a következő megoldáshoz: helyezzünk el i futót a fehér mezőkre és $k - i$ futót a fekete mezőkre, úgy, hogy a fehér mezőre elhelyezett futók nem ütik egymást és a fekete mezőkre elhelyezett futók sem. Minden i-re összeszorozzuk a fehér és a fekete futók elrendezésének számát, majd ezeket a szorzatokat összeadjuk minden $i = 0 \ldots k$-ra.

A feladat megoldható hatékonyabb algoritmusokkal is, de elegendő a fenti módszerrel előre generálni a lehetséges 8281 konfigurációra az eredményeket, ami pár perc alatt elvégezhető.

4.4.3. Zip

A feladatot két részre lehet osztani. Először felépítünk egy n csúcsú súlyozott irányított teljes gráfot, amelyben egy ív súlya azt jelöli, hogy legtöbb hány karakter lehet közös, ha a csúcsoknak megfelelő szavakat egymás után írjuk. A gráf felépítése után meg kell határoznunk a leghosszabb olyan út hosszát, amely pontosan m csúcsot tartalmaz. A keresett szám ($m \cdot k$ - ennek az útnak a hossza) lesz.

Az első rész megoldható a naív algoritmussal $O(n^2 \cdot k^2)$ időben, vagy a *Knuth-Morris-Pratt algoritmus* [25] prefix függvényét használva $O(n^2 \cdot k)$ időben.

A második rész megoldható visszalépéses kereséssel $O(n^m)$ időben, ami természetesen túl lassú a lehetséges optimizálások bevetésével is. Egy hatékonyabb módszer a dinamikus programozás alkalmazása

[2, 3], melynek bonyolultsága $O(n^2 \cdot m)$: kiszámoljuk egy mátrixban a leghosszabb olyan út hosszát, amely adott számú csúcson halad át és egy adott csúcsban végződik. A keresett érték a mátrix utolsó sorában szereplő legnagyobb szám.

4.5. Második feladatsor

4.5.1. Összeg

A feladatot a visszalépéses keresés módszerével oldjuk meg, de a maximális pontszám eléréséhez számos optimizálási lehetőséget kell „bevetnünk". Először is észrevesszük, hogy nincs értelme az eddig megtalált minimális összegnél nagyobb összegeket generálnunk.

A második lépésben tároljuk a téglalap összes eddig fel nem használt nem-nulla elemét rendezve és minden lépésben kiszámítjuk azt az összeget, amelyet legoptimistább esetben elérhetünk. Vagyis ha a k. lépésben vagyunk, akkor $k+1$-től $\frac{n \cdot m}{2}$-ig a lépésszámokat összeszorozzuk a megmaradt elemekkel csökkenő sorrendben, majd ezeket összeadjuk az eddig kialakított összeggel. Ha az így kapott legoptimistább összeg nagyobb vagy egyenlő, mint az eddig megtalált minimális összeg, akkor nincs értelme továbblépnünk.

A harmadik lépésben felépítjük a mátrixhoz rendelhető gráf szomszédsági mátrixát, mely két szempontból is hasznos lesz. Egyrészt nem kell mind a nyolc szomszédról ellenőriznünk, hogy nullától különböző elem-e és a mátrixon belül található-e. Másrészt a szomszédsági lista szerepet kap majd a negyedik javításban is.

Az első két optimizálással megbizonyosodtunk róla, hogy nem

építjük tovább az útvonalat, ha annak összege már meghaladta az eddig talált legjobbat, illetve akkor sem, ha a legjobb esetben is meg fogja haladni azt. Viszont az sem mindegy, hogy milyen sorrendben generáljuk a megoldásokat, érdemes a backtracking minden szintjén azokkal az ágakkal kezdeni, amelyek a legígéretesebbek.

Ezért a negyedik lépésben a felépített gráf minden egyes csúcsához tartozó szomszédsági listát rendezzük a szomszédokhoz rendelt elem értéke szerinti csökkenő sorrendben, hogy mindig a nagyobb elemek irányába induljunk. Végül az ötödik lépésben a keresést is a mátrixban található elemek csökkenő sorrendjében indítjuk.

4.5.2. Törpék

Abból kiindulva, hogy két pont meghatároz egy egyenest, az első ötlet, hogy válasszunk ki hat pontot és ellenőrizzük, hogy az ezek által meghatározott egyenesek lefedik-e a többi pontot. Ennek bonyolultsága $O(n^7)$ lenne, ami túl lassú. A fenti megoldás javítható úgy, hogy csak négy pontot választunk ki és azt ellenőrizzük, hogy azok a pontok, melyeket nem fed le a négy pont által meghatározott két egyenes, egyazon egyenesen találhatóak-e. Ennek a bonyolultsága $O(n^5)$, ami továbbra sem elégséges.

Az optimális megoldás lineáris időben fut. Ha az összes megadott pont kollineáris, a feladat triviálissá válik. Ha nem, válasszunk ki három nem-kollineáris pontot, jelöljük ezeket A, B és C-vel. Két esetünk van: ha az AB, BC és AC egyenesek lefedik az összes pontot, megtaláltuk a megoldást. Ha nem, létezik egy D pont, amely nincs rajta egyiken sem a fenti három egyenes közül. Ekkor az alábbi két

esetben létezhet három egyenes, amely lefedi az összes pontot:

1. Két egyenes lefed két-két pontot az A, B, C és D pontok közül és a harmadik egyenes más pontokat fed le. Ennek az esetnek az ellenőrzéséhez kipróbáljuk az összes lehetséges párosítást: $AB + CD$, $AC + BD$, $AD + BC$.

2. Az egyik egyenes lefed két pontot az A, B, C és D közül és a másik két egyenes egy-egy pontot tartalmaz a megmaradottakból. Ezt az esetet a következőféleképpen kezeljük. Kiválasztunk egy lehetséges párosítást, amely meghatároz két egyenest, például az $AB + CD$-t. Ha az ezek által nem-lefedett pontok nem kollineárisak, akkor keresünk egy E pontot, mely nincs rajta a két egyenesen. Ha az összes pont lefedhető három egyenessel, akkor az E pont ezeknek valamelyikén található. Tudván, hogy az E pontot nem tartalmazza az AB egyenes, két lehetőséget kell csak kipróbálnunk: Az $AB + CE$-t és az $AB + DE$-t. A többi párosítást szimmetrikus módon tárgyaljuk.

Megjegyezzük, hogy egy érdekes megközelítés ennek a feladatnak a megoldására a következő probabilisztikus algoritmus. A megengedett futási idő lejártáig folyamatosan kiválasztunk véletlenszerűen négy pontot és ellenőrizzük, hogy az ezek által meghatározott két egyenes által nem-lefedett pontok kollineárisak-e. Ha igen, akkor találtunk egy megoldást, ha nem, akkor újabb négy pontot választunk ki véletlenszerűen. Ez a megoldás a versenyen a pontszám 85%-át érhette volna el.

4.5.3. Kovács János

Először megjegyezzük, hogy azokat a tárgyakat, amelyek a helyükön vannak, törölhetjük, mert nem kell velük foglalkoznunk a továbbiakban. Ezt követően észrevesszük, hogy egy állapotot a $(room, hand, done)$ számhármassal jellemezhetünk, ahol $room = 1 \dots n$ a szoba száma amelyikben János éppen van, $hand = 0 \dots 2^k - 1$ azon tárgyak bináris reprezentációja, amelyeket János éppen visz és $done = 0 \dots 2^k - 1$ azon tárgyak bináris reprezentációja, amelyek már a helyükre kerültek. Építsünk fel egy súlyozatlan irányított gráfot, melyben két állapot közé akkor teszünk egy ívet, ha egyik állapotból átjuthatunk a másikba egyetlen művelet segítségével. Ezzel levezettük a feladatot a minimális út megkeresésére az $(1, 0, 0)$ állapotból az $(n, 0, 2^k - 1)$ állapotba, amelyet szélességi bejárással [28, 27] találhatunk meg. A megoldás $O(n \cdot 2^{2 \cdot k} + m)$ memóriát és $O((n + k) \cdot n \cdot 2^{2 \cdot k})$ időt igényel.

Egy másik megközelítés az eredeti gráf minden csúcspárja között meghatározni a legrövidebb utat a *Roy-Floyd (Floyd-Warshall) algoritmussal* [31, 15, 34], majd generálni az összes lehetséges sorrendjét a tárgyak felvételének és letételének a visszelépéses keresés módszerével.

5. fejezet

Megoldások

5.1. Gráfelméleti feladatok

5.1.1. Project management

```
1 #include <vector>
2 #include <fstream>
3 #include <functional>
4 #include <algorithm>
5
6 #define FOR(i, a, b) for(int i = (a); i <= (b);
      ++i)
7
8 #define IN_FILE "pm.in"
9 #define OUT_FILE "pm.out"
10
```

```
11 using namespace std;
12
13 int n, nowTime, totalTime;
14 vector <int> time, earliest, latest;
15 vector < vector <bool> > g;
16 vector < pair <int, int> > exitTime;
17 vector <bool> was;
18
19 void readData()
20 {
21   ifstream fin(IN_FILE);
22   fin >> n;
23   time.resize(n + 1);
24   FOR(i, 1, n) fin >> time[i];
25
26   g.resize(n + 1, vector<bool> (n + 1) );
27   FOR(i, 1, n)
28   {
29     int count;
30     fin >> count;
31     FOR(j, 1, count)
32     {
33       int x;
34       fin >> x;
35       g[x][i] = true;
36     }
```

```
37    }
38  }
39
40  void df(int node)
41  {
42    was[node] = true;
43    FOR(i, 1, n)
44      if (g[node][i] && !was[i]) df(i);
45    ++nowTime;
46    exitTime[node] = make_pair(nowTime, node);
47  }
48
49  void topSort()
50  {
51    exitTime.resize(n + 1);
52    was.resize(n + 1, false);
53    nowTime = 0;
54    FOR(i, 1, n)
55      if (!was[i]) df(i);
56  }
57
58  void cpm()
59  {
60    earliest.resize(n + 1);
61    totalTime = 0;
62    FOR(i, 1, n)
```

```
63    {
64      int node = exitTime[i].second;
65      earliest[node] = 0;
66      FOR(j, 1, n)
67        if (g[j][node]) earliest[node] = max(
              earliest[node], earliest[j] + time[j]);
68      totalTime = max(totalTime, earliest[node] +
            time[node]);
69    }
70
71    latest.resize(n + 1);
72    for(int i = n; i >= 1; --i)
73    {
74      int node = exitTime[i].second;
75      latest[node] = totalTime - time[node];
76      FOR(j, 1, n)
77        if (g[node][j]) latest[node] = min(latest[
              node], latest[j] - time[node]);
78    }
79  }
80
81  void writeData()
82  {
83    ofstream fout(OUT_FILE);
84    fout << totalTime << endl;
85    FOR(i, 1, n) fout << earliest[i] << " " <<
```

```
        latest[i] << endl;
86  }
87
88  int main()
89  {
90      readData();
91      topSort();
92      sort(exitTime.begin() + 1, exitTime.end(),
            greater< pair <int, int> > ());
93      cpm();
94      writeData();
95      return 0;
96  }
```

5.1.2. Cannibal

```
1  #include <fstream>
2  #include <map>
3  #include <queue>
4  #include <time.h>
5  #include <iostream>
6
7  using namespace std;
8
9  int sol[51][51][100]=//Hely hianyaban nem
       publikaljuk a kiszamolt konstans ertekeket
10
```

```
11 struct Tstate
12 {
13   bool side;
14   int can,mis;
15 };
16
17 bool operator< (const Tstate &a, const Tstate &b)
18 {
19   return a.side < b.side || a.side == b.side && a
       .can < b.can ||
20     a.side == b.side && a.can == b.can && a.mis <
         b.mis;
21 }
22
23 int n, m, k;
24 map <Tstate,int> levels;
25 queue <Tstate> qStates;
26
27 int bf() //Breadth first search
28 {
29   if (n > m) return -1;
30   if (k >= n + m) return (n + m > 0);
31   while (!qStates.empty()) qStates.pop();
32   levels.clear();
33   Tstate aux, final;
34   //Starting state
```

```
35    aux.side = false;
36    aux.can = n;
37    aux.mis = m;
38    //Final state
39    final.can = 0;
40    final.mis = 0;
41    final.side = true;
42    qStates.push(aux);
43    levels[aux] = 0;
44    while (!qStates.empty() && !levels.count(final)
          )
45    {
46      Tstate head = qStates.front();
47      int aCan, aMis; //The number of cannibals and
            missionaries on the current side
48      if (!head.side)
49      {
50        aCan = head.can;
51        aMis = head.mis;
52      }
53      else
54      {
55        aCan = n-head.can;
56        aMis = m-head.mis;
57      }
58      for (int q = 0; q <= min(k,aMis); ++q)
```

```
59          //The number of missionaries to travel with
                the boat
60          //It must be less or equal with:
61          // -the number of missionaries on the
               current side of the river
62          // -the size of the boat
63        {
64          for (int w = 0; w <= min(k-q,aCan);++w)
65            //The number of cannibals to travel with
                 the boat
66            //It must be less or equal with:
67            // -the number of missionaries to travel
                 with the boat (if it's >0)
68            // -the remaining places in the boat
69            // -the number of cannibals on the
                 current side of the river
70          {
71            if (q + w == 0) continue; //Somebody must
                  be in the boat
72            if (q && w > q) break;
73            aux.side = !head.side;
74            if (!head.side)
75            {
76              aux.can = head.can - w;
77              aux.mis = head.mis - q;
78            }
```

```
79          else
80          {
81            aux.can = head.can + w;
82            aux.mis = head.mis + q;
83          }
84          if ((aux.can <= aux.mis || aux.mis == 0)
              &&
85            ((n - aux.can) <= (m-aux.mis) || aux.
                mis == m) &&
86            //The property must be maintained on
                both sides
87            //otherwise the cannibals will eat the
                missionaries
88            //If there are 0 missionaries, there's
                nothing to eat
89            !levels.count(aux)) //State not visited
                 yet
90          {
91            levels[aux] = levels[head] + 1;
92            qStates.push(aux);
93          }
94        }
95      }
96      qStates.pop();
97    }
98    if (levels.count(final)) return levels[final];
```

```
 99    else return -1;
100 }
101
102
103 void main()
104 {
105    ifstream fin("cannibal.in");
106    ofstream fout("cannibal.out");
107    do
108    {
109       fin >> n >> m >> k;
110       if (n == -1) break;
111       if (n <= 50 && m <= 50 && k <= 100) fout <<
             sol[n][m][k-1] << endl;
112       else fout << bf() << endl;
113    } while (1);
114    fin.close();
115    fout.close();
116 }
```

5.1.3. Domino2

```
 1 #include <vector>
 2 #include <fstream>
 3 #include <stack>
 4
 5 #define FOR(i, a, b) for(int i = (a); i <= (b);
```

```
    ++i)
6
7  #define IN_FILE "domino2.in"
8  #define OUT_FILE "domino2.out"
9
10 using namespace std;
11
12 int n;
13 vector < vector <int> > g(10, vector <int> (10));
14 vector < vector < stack <int> > > input(10,
       vector < stack <int> > (10, stack <int> ()));
15 vector <int> path, grade(10);
16
17 void readData()
18 {
19   ifstream fin(IN_FILE);
20   fin >> n;
21   FOR(i, 1, n)
22   {
23     int x, y;
24     fin >> x >> y;
25     input[x][y].push(i);
26     ++g[x][y];
27     ++g[y][x];
28     ++grade[x];
29     ++grade[y];
```

```
30    }
31  }
32
33  void euler(int node)
34  {
35    FOR(i, 0, 9)
36      if (g[node][i])
37      {
38        --g[node][i];
39        --g[i][node];
40        euler(i);
41      }
42    path.push_back(node);
43  }
44
45  void writeData()
46  {
47    ofstream fout(OUT_FILE);
48    if (path.size() != n + 1)
49    {
50      fout << 0;
51      return;
52    }
53    fout << 1 << endl;
54    FOR(i, 0, path.size() - 2)
55    {
```

```
56        int x = path[i];
57        int y = path[i + 1];
58        if (!input[x][y].empty())
59        {
60          fout << input[x][y].top() << " 0\n";
61          input[x][y].pop();
62        }
63        else
64        {
65          fout << input[y][x].top() << " 1\n";
66          input[y][x].pop();
67        }
68      }
69  }
70
71  int main()
72  {
73      readData();
74      int odd = -1;
75      FOR(i, 0, 9)
76        if (grade[i] & 1)
77        {
78          odd = i;
79          break;
80        }
81      if (odd == -1)
```

```
82      FOR(i, 0, 9)
83        if (grade[i])
84        {
85          odd = i;
86          break;
87        }
88    euler(odd);
89    writeData();
90    return 0;
91  }
```

5.1.4. Online

```
1  #include <vector>
2  #include <fstream>
3  #include <list>
4
5  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
6  #define X first
7  #define Y second
8
9  #define IN_FILE "online.in"
10 #define OUT_FILE "online.out"
11
12 #define INF 300
13
```

```
14 using namespace std;
15
16 int n, m, k, treeCost = 0;
17 vector < vector < pair <int, int> > > g;
18 vector < list < pair <int, int> > > tree;
19 vector < pair < pair < int, int >, int > > edges;
20 vector <bool> was;
21 vector <int> d, father;
22 vector < pair <int, int> > ss;
23 vector < vector < list< pair <int, int> > ::
      iterator> > treePointers;
24
25 ofstream fout(OUT_FILE);
26
27 void readData()
28 {
29   ifstream fin(IN_FILE);
30   fin >> n >> m;
31   g.resize(n + 1);
32   FOR(i, 1, m)
33   {
34     int x, y, cost;
35     fin >> x >> y >> cost;
36     g[x].push_back(make_pair(y, cost));
37     g[y].push_back(make_pair(x, cost));
38   }
```

```
39    fin >> k;
40    edges.resize(k + 1);
41    FOR(i, 1, k)
42      fin >> edges[i].X.X >> edges[i].X.Y >> edges[
          i].Y;
43  }
44
45  void addToTree(int x, int y, int cost)
46  {
47    tree[x].push_back(make_pair(y, cost));
48    treePointers[x][y] = --tree[x].end();
49    tree[y].push_back(make_pair(x, cost));
50    treePointers[y][x] = --tree[y].end();
51  }
52
53  void removeFromTree(int x, int y)
54  {
55    tree[x].erase(treePointers[x][y]);
56    tree[y].erase(treePointers[y][x]);
57  }
58
59  void prim()
60  {
61    was.resize(n + 1);
62    father.resize(n + 1);
63    tree.resize(n + 1);
```

```
64    treePointers.resize(n + 1, vector < list< pair
        <int, int> > :: iterator> (n + 1));
65    d.resize(n + 1, INF);
66    was[1] = true;
67    FOR(i, 0, g[1].size() - 1)
68    {
69      pair <int, int> edge = g[1][i];
70      if (d[edge.X] > edge.Y)
71      {
72        d[edge.X] = edge.Y;
73        father[edge.X] = 1;
74      }
75    }
76    FOR(i, 2, n)
77    {
78      int node = -1;
79      FOR(j, 2, n)
80        if (!was[j] && (node == -1 || d[node] > d[j
            ])) node = j;
81      was[node] = true;
82      treeCost += d[node];
83      addToTree(node, father[node], d[node]);
84      FOR(j, 0, g[node].size() - 1)
85      {
86        pair <int, int> edge = g[node][j];
87        if (!was[edge.X] && d[edge.X] > edge.Y)
```

```
88          {
89            d[edge.X] = edge.Y;
90            father[edge.X] = node;
91          }
92        }
93      }
94  }
95
96  void df(int node, int sp, int dest)
97  {
98    was[node] = true;
99    if (node == dest) return;
100   for (list <pair <int, int> >::iterator it =
          tree[node].begin(); it != tree[node].end();
          ++it)
101   {
102     pair <int, int> edge = *it;
103     if (!was[edge.X])
104     {
105       ss[sp + 1].X = edge.X;
106       ss[sp + 1].Y = edge.Y;
107       df(edge.X, sp + 1, dest);
108       if (was[dest]) return;
109     }
110   }
111 }
```

```
112
113 void solve()
114 {
115   ss.resize(n + 1);
116   FOR(i, 1, k)
117   {
118     fill(was.begin(), was.end(), false);
119     ss[1].X = edges[i].X.X;
120     ss[1].Y = 0;
121     df(edges[i].X.X, 1, edges[i].X.Y);
122     int maxCost = 0, x, y;
123     FOR(j, 2, n)
124     {
125       if (ss[j].Y > maxCost)
126       {
127         maxCost = ss[j].Y;
128         x = ss[j].X;
129         y = ss[j - 1].X;
130       }
131       if (ss[j].X == edges[i].X.Y) break;
132     }
133     if (maxCost > edges[i].Y)
134     {
135       removeFromTree(x, y);
136       addToTree(edges[i].X.X, edges[i].X.Y, edges
              [i].Y);
```

```
137         treeCost = treeCost - maxCost + edges[i].Y;
138       }
139       fout << treeCost << '\n';
140     }
141 }
142
143 int main()
144 {
145     readData();
146     prim();
147     fout << treeCost << '\n';
148     solve();
149     return 0;
150 }
```

5.2. Dinamikus programozás módszerével megoldható feladatok

5.2.1. Persely

```
1 #include <vector>
2 #include <fstream>
3
4 #define FOR(i, a, b) for(int i = (a); i <= (b);
      ++i)
5
```

```
6  #define IN_FILE "p.in"
7  #define OUT_FILE "p.out"
8
9  #define INF 2000000
10
11 using namespace std;
12
13 int s, n;
14 vector <int> weight, value, dp;
15
16 void readData()
17 {
18   ifstream fin(IN_FILE);
19   fin >> s >> n;
20   weight.resize(n + 1);
21   value.resize(n + 1);
22   FOR(i, 1, n) fin >> weight[i] >> value[i];
23 }
24
25 void writeData()
26 {
27   ofstream fout(OUT_FILE);
28   fout << dp[s];
29 }
30
31 int main()
```

```
32 {
33   readData();
34   dp.resize(s + 1, INF);
35   dp[0] = 0;
36   FOR(i, 1, s)
37     FOR(j, 1, n)
38       if (i >= weight[j] && dp[i] > dp[i - weight
           [j]] + value[j])
39         dp[i] = dp[i - weight[j]] + value[j];
40
41   writeData();
42   return 0;
43 }
```

5.2.2. Domino

```
1 #include <vector>
2 #include <fstream>
3
4 #define FOR(i, a, b) for(int i = (a); i <= (b);
    ++i)
5
6 #define IN_FILE "domino.in"
7 #define OUT_FILE "domino.out"
8
9 using namespace std;
10
```

```
11 int n, sol = 1;
12 vector <vector < vector <int > > > dp(2, vector <
       vector <int> > (10, vector <int> (10)));
13
14 void readData()
15 {
16   ifstream fin(IN_FILE);
17   fin >> n;
18   int act = 0, next = 1;
19   int x, y;
20   fin >> x >> y;
21   dp[act][x][y] = 1;
22   FOR(i, 2, n)
23   {
24     fin >> x >> y;
25     FOR(j, 0, 9)
26     {
27       if (dp[act][j][x] && dp[next][j][y] < dp[
           act][j][x] + 1)
28         dp[next][j][y] = dp[act][j][x] + 1;
29       if (dp[act][j][y] && dp[next][j][x] < dp[
           act][j][y] + 1)
30         dp[next][j][x] = dp[act][j][y] + 1;
31       if (dp[act][x][j] && dp[next][y][j] < dp[
           act][x][j] + 1)
32         dp[next][y][j] = dp[act][x][j] + 1;
```

```
33          if (dp[act][y][j] && dp[next][x][j] < dp[
                act][y][j] + 1)
34            dp[next][x][j] = dp[act][y][j] + 1;
35
36        }
37        sol = 0;
38        FOR(x, 0, 9)
39          FOR(y, 0, 9)
40          {
41            dp[next][x][y] = max(dp[next][x][y], dp[
                  act][x][y]);
42            sol = max(sol, dp[next][x][y]);
43          }
44
45        act = 1 - act;
46        next = 1 - next;
47    }
48
49 }
50
51 void writeData()
52 {
53    ofstream fout(OUT_FILE);
54    fout << sol;
55 }
56
```

```
57 int main()
58 {
59
60   readData();
61   writeData();
62   return 0;
63 }
```

5.2.3. Tango

Az alábbi megoldásban egy frázis eltáncolási lehetőségeit a dinamikus programozás módszerével határozzuk meg.

```
 1 #include <vector>
 2 #include <fstream>
 3
 4 using namespace std;
 5
 6 #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
 7
 8 #define IN_FILE "tango.in"
 9 #define OUT_FILE "tango.out"
10 #define BASE 999983
11
12 ifstream fin(IN_FILE);
13 ofstream fout(OUT_FILE);
14
```

```
15 int n, k;
16 vector <int> nr(9), dp(9);
17
18 void Knapsack()
19 {
20   dp[0] = 1;
21   FOR(i, 1, 8)
22     FOR(j, 1, i) dp[i] = (dp[i] + (long long) nr[
         j] * dp[i - j]) % BASE;
23 }
24
25 int Solve()
26 {
27   int exp = k / 8;
28   long long ans = 1, pow2 = dp[8];
29   while (exp)
30   {
31     if (exp & 1) ans = (ans * pow2) % BASE;
32     pow2 = (pow2 * pow2) % BASE;
33     exp >>= 1;
34   }
35   return (ans * dp[k % 8]) % BASE;
36 }
37
38 int main()
39 {
```

```
40     //Read data
41     fin >> n >> k;
42     FOR(i, 1, n)
43     {
44       int x;
45       fin >> x;
46       if (x <= 8) ++nr[x];
47     }
48     fin.close();
49
50     //Solve
51     Knapsack();
52
53     //Write data
54     fout << Solve();
55     fout.close();
56
57     return 0;
58 }
```

5.2.4. Dday

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <vector>
4 #include <algorithm>
```

```
5
6  using namespace std;
7
8  #define MAX 101
9
10 int n, m, k;
11 long dom[MAX], c[MAX][MAX][MAX];
12 int ppl[MAX], money[MAX], father[MAX][MAX][MAX];
13 vector < int > ans;
14
15 void readData()
16 {
17   FILE *fin = fopen("dday.in", "r");
18   fscanf(fin, "%d%d%d", &n, &m, &k);
19   for (int q = 0; q < n; ++q)
20     fscanf(fin, "%ld%d%d", &dom[q], &money[q], &
         ppl[q]);
21   fclose(fin);
22 }
23
24 void solve()
25 {
26   for (int i = 0; i <= m; ++i)
27   {
28     for (int j = 0; j <= k; ++j) c[0][i][j] = -1;
29   }
```

```
30    c[0][0][0] = 0;
31
32    for (int q = 0; q < n; ++q)
33    {
34      for (int i = 0; i <= m; ++i)
35      {
36        for (int j = 0; j <= k; ++j)
37        {
38          c[q + 1][i][j] = c[q][i][j];
39          father[q + 1][i][j] = father[q][i][j];
40        }
41      }
42      for (int i = 0; i <= m - money[q]; ++i)
43      {
44        for (int j = 0; j <= k - ppl[q]; ++j)
45        {
46          if (c[q][i][j] != -1 && c[q + 1][i +
              money[q]][j + ppl[q]] < c[q][i][j] +
              dom[q])
47          {
48            c[q + 1][i + money[q]][j + ppl[q]] = c[
                q][i][j] + dom[q];
49            father[q + 1][i + money[q]][j + ppl[q]]
                  = q;
50          }
51        }
```

```
52      }
53    }
54  }
55
56  void writeData()
57  {
58    int besti = 0,bestj = 0;
59    for(int i = 1; i <= m; ++i)
60    {
61      for(int j = 1; j <= k; ++j)
62      {
63        if (c[n][besti][bestj] < c[n][i][j])
64        {
65          besti = i;
66          bestj = j;
67        }
68      }
69    }
70
71    int i = besti;
72    int j = bestj;
73    int q = n;
74    while (i > 0)
75    {
76      int aux = father[q][i][j];
77      ans.push_back(aux + 1);
```

```
78        i -= money[aux];
79        j -= ppl[aux];
80        q = aux;
81      }
82      sort(ans.begin(), ans.end());
83
84      FILE *fout = fopen("dday.out","w");
85      fprintf(fout, "%d␣%ld␣%d␣%d\n", ans.size(), c[n
          ][besti][bestj], besti, bestj);
86      for(q = 0; q < ans.size(); ++q) fprintf(fout, "
          %d␣", ans[q]);
87      fclose(fout);
88  }
89
90
91
92  void main()
93  {
94      readData();
95      solve();
96      writeData();
97  }
```

5.3. Adatszerkezetekkel kapcsolatos feladatok

5.3.1. Raktár

```
1  #include <vector>
2  #include <string>
3  #include <fstream>
4  #include <sstream>
5
6  using namespace std;
7
8  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
9
10 #define IN_FILE "raktar.in"
11 #define OUT_FILE "raktar.out"
12
13 ifstream fin(IN_FILE);
14 ofstream fout(OUT_FILE);
15
16 vector<int> arak(500000);
17
18 void Termek(string nev, int ar)
19 {
20   int i = 0;
```

```
21    FOR(j, 0, 3) i = i * 26 + (nev[j] - 'a');
22    arak[i] = ar;
23  }
24
25  int Ar(string nev)
26  {
27    int i = 0;
28    FOR(j, 0, 3) i = i * 26 + (nev[j] - 'a');
29    return arak[i];
30  }
31
32  int main()
33  {
34    string muvelet;
35    while (fin >> muvelet)
36    {
37      if (muvelet.substr(0, 6) == "TERMEK")
38      {
39        stringstream nr(muvelet.substr(12, muvelet.
              size() - 13));
40        int ar;
41        nr >> ar;
42        string kod = muvelet.substr(7, 4);
43        Termek(kod, ar);
44      }
45      else
```

```
46      {
47        string kod = muvelet.substr(3, 4);
48        fout << Ar(kod) << endl;
49      }
50    }
51    fin.close();
52    fout.close();
53
54    return 0;
55  }
```

5.3.2. Ékszerek

```
1  #include <vector>
2  #include <fstream>
3  #include <sstream>
4
5  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
6
7  #define IN_FILE "ekszer.in"
8  #define OUT_FILE "ekszer.out"
9
10 using namespace std;
11
12 int heapSize = 0;
13 vector <int> heap(1 << 15);
```

```
14
15 void up(int index)
16 {
17   int aux = heap[index];
18   while (index > 1 && aux > heap[index / 2])
19   {
20     heap[index] = heap[index / 2];
21     index /= 2;
22   }
23   heap[index] = aux;
24 }
25
26 void down(int index)
27 {
28   int aux = heap[index];
29   while (1)
30   {
31     int next = -1;
32     if (heapSize >= index * 2) next = index * 2;
33     if (heapSize >= index * 2 + 1 && heap[next] <
         heap[index * 2 + 1]) next = index * 2 +
        1;
34     if (next == -1 || aux >= heap[next]) break;
35     heap[index] = heap[next];
36     index = next;
37   }
```

```
38    heap[index] = aux;
39  }
40
41  void add(int value)
42  {
43    ++heapSize;
44    heap[heapSize] = value;
45    up(heapSize);
46  }
47
48  int remove()
49  {
50    int ans = heap[1];
51    swap(heap[1], heap[heapSize]);
52    --heapSize;
53    down(1);
54    return ans;
55  }
56
57  int main()
58  {
59    ifstream fin(IN_FILE);
60    ofstream fout(OUT_FILE);
61    string s;
62    while (fin >> s)
63    {
```

```
64        if (s.substr(0, 4) == "VEVO")
65        {
66          fout << remove() << endl;
67        }
68        else
69        {
70          stringstream ss(s.substr(7, s.size() - 7));
71          int ar;
72          ss >> ar;
73          add(ar);
74        }
75      }
76      return 0;
77  }
```

5.3.3. Cég

Az alábbi megoldás intervallumfákat használ.

```
1  #include <vector>
2  #include <fstream>
3  #include <string>
4  #include <sstream>
5
6  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
7
8  #define IN_FILE "ceg.in"
```

```
 9 #define OUT_FILE "ceg.out"
10
11 using namespace std;
12
13 int n;
14 vector <int> ai(8192);
15
16 void update(int index, int dif, int node, int
       left, int right)
17 {
18   if (left == right)
19   {
20     ai[node] += dif;
21     return;
22   }
23   int mid = (left + right) / 2;
24   if (index <= mid) update(index, dif, node * 2,
       left, mid);
25   else update(index, dif, node * 2 + 1, mid + 1,
       right);
26   ai[node] = ai[node * 2] + ai[node * 2 + 1];
27 }
28
29 int query(int ind1, int ind2, int node, int left,
     int right)
30 {
```

```
31    if (ind1 <= left && right <= ind2) return ai[
          node];
32    int mid = (left + right) / 2;
33    int ans = 0;
34    if (ind1 <= mid) ans += query(ind1, ind2, node
          * 2, left, mid);
35    if (mid + 1 <= ind2) ans += query(ind1, ind2,
          node * 2 + 1, mid + 1, right);
36    return ans;
37  }
38
39  int main()
40  {
41    ifstream fin(IN_FILE);
42    ofstream fout(OUT_FILE);
43    fin >> n;
44    string s;
45    while (fin >> s)
46    {
47      int ind1 = s.find('('), ind2 = s.find(','),
            ind3 = s.find(')');
48      int p1, p2;
49      stringstream ss(s.substr(ind1 + 1, ind2 -
            ind1 - 1));
50      stringstream ss2(s.substr(ind2 + 1, ind3 -
            ind2 - 1));
```

```
51      ss >> p1;
52      ss2 >> p2;
53
54      if (s.substr(0, 8) == "JELENTES")
55      {
56        update(p1, p2, 1, 1, n);
57      }
58      else
59      {
60        fout << query(p1, p2, 1, 1, n) << endl;
61      }
62    }
63    return 0;
64 }
```

5.3.4. Motel

```
1 #include <vector>
2 #include <fstream>
3 #include <algorithm>
4
5 #define FOR(i, a, b) for(int i = (a); i <= (b);
      ++i)
6 #define X first
7 #define Y second
8
```

```
 9 #define IN_FILE "motel.in"
10 #define OUT_FILE "motel.out"
11
12 using namespace std;
13
14 int n;
15 vector < pair <pair <int, int>, int > > a;
16 vector < pair <int, int> > b, sol;
17 int heapSize = 0;
18 vector <int> heap(8192);
19
20 ofstream fout(OUT_FILE);
21
22 void readData()
23 {
24   ifstream fin(IN_FILE);
25   fin >> n;
26   a.resize(n + 1);
27   b.resize(n + 1);
28   FOR(i, 1, n)
29   {
30     fin >> a[i].X.X >> a[i].X.Y;
31     a[i].Y = i;
32   }
33   FOR(i, 1, n)
34   {
```

```
35      fin >> b[i].X;
36      b[i].Y = i;
37    }
38  }
39
40  void up(int index)
41  {
42    int aux = heap[index];
43    while (index > 1 && a[aux].X.Y < a[heap[index /
          2]].X.Y)
44    {
45      heap[index] = heap[index / 2];
46      index /= 2;
47    }
48    heap[index] = aux;
49  }
50
51  void down(int index)
52  {
53    int aux = heap[index];
54    while (1)
55    {
56      int next = -1;
57      if (heapSize >= index * 2) next = index * 2;
58      if (heapSize >= index * 2 + 1 && a[heap[next
          ]].X.Y > a[heap[index * 2 + 1]].X.Y) next
```

```
        = index * 2 + 1;
59      if (next == -1 || a[aux].X.Y <= a[heap[next
        ]].X.Y) break;
60      heap[index] = heap[next];
61      index = next;
62    }
63    heap[index] = aux;
64  }
65
66  void add(int value)
67  {
68    ++heapSize;
69    heap[heapSize] = value;
70    up(heapSize);
71  }
72
73  int remove()
74  {
75    int ans = heap[1];
76    swap(heap[1], heap[heapSize]);
77    --heapSize;
78    down(1);
79    return ans;
80  }
81
82
```

```
83 void solve()
84 {
85   sort(a.begin() + 1, a.end());
86   sort(b.begin() + 1, b.end());
87   int j = 1;
88   FOR(i, 1, n)
89   {
90     while (j <= n && a[j].X.X <= b[i].X)
91     {
92       add(j);
93       ++j;
94     }
95     while (heapSize > 0 && a[heap[1]].X.Y < b[i].
         X)
96       remove();
97     if (!heapSize)
98     {
99       fout << "0 0";
100      return;
101    }
102    sol.push_back(make_pair(a[heap[1]].Y, b[i].Y)
         );
103    remove();
104  }
105 }
106
```

```
107 int main()
108 {
109   readData();
110   solve();
111   if (sol.size() == n)
112     FOR(i, 0, n - 1) fout << sol[i].X << " " <<
          sol[i].Y << '\n';
113   return 0;
114 }
```

5.4. Első feladatsor

5.4.1. Tekaj

```
1 #include <stdio.h>
2 #include <string.h>
3
4 #define MAX 200000
5
6 int n, i;
7 int a[MAX], was[MAX];
8
9 void main()
10 {
11   FILE *fin = fopen("tekaj.in", "r");
12
13   fscanf(fin, "%ld", &n);
```

```
14    for (i = 0; i < n; ++i) fscanf(fin,"%d",&a[i]);
15    fclose(fin);
16
17    memset(was, 0, sizeof(was));
18    int aux;
19    for(i=0; i<n; ++i)
20    {
21      aux=a[i];
22      while (!(aux&1)) aux >>= 1;
23      if (!was[aux]) was[aux] = i;
24      else break;
25    }
26
27    FILE *fout = fopen("tekaj.out", "w");
28    if (a[i] > a[was[aux]]) fprintf(fout, "1\n%d\n1
        \n%d", i + 1, was[aux] + 1);
29    else fprintf(fout, "1\n%d\n1\n%d", was[aux]+1,
        i+1);
30    fclose(fout);
31  }
```

5.4.2. Bishop

Az alábbi program felhasználható a megoldások generálására.

```
1 #include <fstream>
2 #include <vector>
3 #include <time.h>
```

```
4
5  using namespace std;
6
7  int n,m,k;
8  long long sol,ans;
9  vector < vector < pair<int ,int > > > diag;
10 vector <int> ss, ss2;
11 vector <bool> was;
12
13 void BuildDiagonals()
14 {
15   diag.clear();
16   diag.resize(n+m-1);
17   for (int q=2;q<=min(n,m);++q)
18   {
19     for (int w=1;w<q;++w)
20     {
21       diag[q-2].push_back(make_pair(w,q-w));
22     }
23   }
24   for (int q=min(n,m);q<=max(n,m);++q)
25   {
26     for (int w=1;w<=min(m,n);++w)
27     {
28       if (n>m) diag[q-1].push_back(make_pair(w+q-
             min(n,m),min(n,m)-w+1));
```

```
29          else diag[q-1].push_back(make_pair(min(n,m)
                -w+1,w+q-min(n,m)));
30        }
31      }
32      for (int q=2;q<=min(n,m);++q)
33      {
34        for (int w=1;w<q;++w)
35        {
36          diag[n+m-q].push_back(make_pair(n-w+1,m-(q-
                w)+1));
37        }
38      }
39  }
40
41  void back2(int sp2, int nDiag)//Brute force the
        bishops on the selected diagonals
42  {
43    if (sp2==nDiag) ++sol;
44    else
45      for (ss2[sp2]=0;ss2[sp2]<diag[ss[sp2+1]].size
            ();++ss2[sp2])
46      {
47        pair <int ,int> aDiag=diag[ss[sp2+1]][ss2[
              sp2]];
48        if (!was[aDiag.first-aDiag.second+m-1])
49        {
```

```
50            was[aDiag.first-aDiag.second+m-1]=true;
51            back2(sp2+1, nDiag);
52            was[aDiag.first-aDiag.second+m-1]=false;
53          }
54        }
55  }
56
57  void back(int sp, int nDiag)//Select nDiag
        diagonals
58  {
59    if (sp>nDiag)
60    {
61      was.clear();
62      was.resize(n+m-1,false);
63      back2(0, nDiag);
64    }
65    else
66      for (ss[sp]=ss[sp-1]+2;ss[sp]<n+m-1-nDiag+sp
          ;++++ss[sp]) back(sp+1, nDiag);
67  }
68
69  void solve()
70  {
71    int even;
72    for (int i=0;i<=k;++i)
73    {
```

```
74        ss.resize(i+1);
75        ss2.resize(i);
76        ss[0]=-2;
77        sol=0;
78        if (i) back(1,i);//Brute force placing of i
              bishops on white squares
79        else sol=1;
80        even=sol;
81        ss.resize(k-i+1);
82        ss2.resize(k-i);
83        ss[0]=-1;
84        sol=0;
85        if (k-i) back(1,k-i);//Brute force placing of
              k-i bishops on black squares
86        else sol=1;
87        ans+=sol*even;
88     }
89  }
90
91  void main()
92  {
93     ifstream fin("bishop.in");
94     ofstream fout("bishop.out");
95     do
96     {
97        fin>>n>>m>>k;
```

```
98        if (n==-1) break;
99        BuildDiagonals();
100       ans=0;
101       solve();
102       fout<<ans<<endl;
103     } while (1);
104     fin.close();
105     fout.close();
106 }
```

5.4.3. Zip

```
1  #include <vector>
2  #include <fstream>
3  #include <string>
4
5  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
6
7  #define IN_FILE "zip.in"
8  #define OUT_FILE "zip.out"
9
10 using namespace std;
11
12 int n, m, k;
13 vector <string> a;
```

```
14 vector < vector <int> > g, dp;
15 vector <int> prefix;
16
17 void readData()
18 {
19   ifstream fin(IN_FILE);
20   fin >> n >> m >> k;
21   a.resize(n + 1);
22   FOR(i, 1, n) fin >> a[i];
23 }
24
25 int getPrefix(string s)
26 {
27   prefix[1] = 0;
28   int i = 2;
29   int len = 0;
30   while (i <= 2 * k)
31   {
32     if (s[i] == s[len + 1])
33     {
34       ++len;
35       prefix[i] = len;
36       ++i;
37     }
38     else
39       if (len > 0)
```

```
40          len = prefix[len];
41        else
42        {
43          prefix[i] = 0;
44          ++i;
45        }
46    }
47    int ans = prefix[2 * k];
48    return ans < k ? ans : prefix[k];
49  }
50
51  void buildGraph()
52  {
53    prefix.resize(2 * k + 1);
54    g.resize(n + 1, vector<int>(n + 1));
55    FOR(i, 1, n)
56      FOR(j, 1, n)
57        g[i][j] = getPrefix("#" + a[j] + a[i]);
58  }
59
60  void solve()
61  {
62    dp.resize(m + 1, vector <int> (n + 1));
63    FOR(i, 2, m)
64      FOR(j, 1, n)
65        FOR(k, 1, n)
```

```
66          dp[i][j] = max(dp[i][j], dp[i - 1][k] + g
                [k][j]);
67  }
68
69  void writeData()
70  {
71    ofstream fout(OUT_FILE);
72    int ans = 0;
73    FOR(i, 1, n) ans = max(ans, dp[m][i]);
74    fout << m * k - ans;
75  }
76
77  int main()
78  {
79    readData();
80    buildGraph();
81    solve();
82    writeData();
83    return 0;
84  }
```

5.5. Második feladatsor

5.5.1. Összeg

```
1  #include <vector>
2  #include <fstream>
```

```
3  #include <set>
4  #include <algorithm>
5  #include <functional>
6
7  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
8  #define X first
9  #define Y second
10
11 #define IN_FILE "osszeg.in"
12 #define OUT_FILE "osszeg.out"
13
14 using namespace std;
15
16 int n, m, minSum = 65000, len;
17 vector < vector <int> > a;
18 vector < vector < vector < pair <int, pair< int,
       int > > > > > g;
19 vector < pair <int, pair <int, int> > > order;
20 vector < vector <bool> > was;
21 vector <int> route, solution;
22 multiset <int> values;
23 int dx[8] = {-1, -1, -1, 0, 0, 1, 1, 1};
24 int dy[8] = {-1, 0, 1, -1, 1, -1, 0, 1};
25
26 void readData()
```

```
27 {
28   ifstream fin(IN_FILE);
29   fin >> n >> m;
30   a.resize(n + 2, vector <int> (m + 2));
31   g.resize(n + 1, vector < vector < pair < int,
        pair <int, int> > > > (m + 1));
32   FOR(i, 1, n)
33     FOR(j, 1, m)
34     {
35       fin >> a[i][j];
36       if (a[i][j])
37       {
38         values.insert(a[i][j]);
39         order.push_back(make_pair(a[i][j],
             make_pair(i, j)));
40       }
41     }
42   sort(order.begin(), order.end(), greater<pair <
       int, pair <int, int> > >());
43
44   FOR(i, 1, n)
45     FOR(j, 1, m)
46       if (a[i][j])
47       {
48         FOR(l, 0, 7)
49         {
```

```
50              int newX = i + dx[l];
51              int newY = j + dy[l];
52              if (a[newX][newY]) g[i][j].push_back(
                    make_pair(a[newX][newY], make_pair(
                    newX, newY)));
53            }
54            sort(g[i][j].begin(), g[i][j].end(),
                  greater<pair <int, pair <int, int> >
                  >());
55          }
56  }
57
58  void back(int row, int col, int sum, int k)
59  {
60    if (!a[row][col] || was[row][col]) return;
61    if (sum + k * a[row][col] >= minSum) return;
62
63    route[k] = a[row][col];
64    if (k == len)
65    {
66      minSum = sum + k * a[row][col];
67      solution = route;
68      return;
69    }
70
71    values.erase(values.find(a[row][col]));
```

```
72    int bestSum = sum + k * a[row][col];
73    int i = len;
74    for (multiset<int>::iterator it = values.begin
          (); i >= k + 1; ++it, --i)
75      bestSum += i * (*it);
76    if (bestSum >= minSum)
77    {
78      values.insert(a[row][col]);
79      return;
80    }
81
82    was[row][col] = true;
83    FOR(i, 0, g[row][col].size() - 1)
84    {
85      pair <int, pair <int, int> > next = g[row][
            col][i];
86      back(next.Y.X, next.Y.Y, sum + k * a[row][col
            ], k + 1);
87    }
88    was[row][col] = false;
89    values.insert(a[row][col]);
90  }
91
92  void writeData()
93  {
94    ofstream fout(OUT_FILE);
```

```
95    fout << minSum << endl;
96    FOR(i, 1, len) fout << solution[i] << " ";
97  }
98
99  int main()
100 {
101   readData();
102
103   len = n * m / 2;
104   route.resize(len + 1);
105   was.resize(n + 1, vector <bool> (m + 1));
106   FOR(i, 0, len - 1) back(order[i].Y.X, order[i].
        Y.Y, 0, 1);
107
108   writeData();
109   return 0;
110 }
```

5.5.2. Törpék

```
1  #include <vector>
2  #include <fstream>
3
4  #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
5  #define X first
6  #define Y second
```

```
 7
 8 #define IN_FILE "torpek.in"
 9 #define OUT_FILE "torpek.out"
10
11 using namespace std;
12
13 int n;
14 vector < pair <int, int> > a;
15
16 void readData()
17 {
18   ifstream fin(IN_FILE);
19   fin >> n;
20   a.resize(n + 1);
21   FOR(i, 1, n) fin >> a[i].X >> a[i].Y;
22 }
23
24 //Are these three points collinear?
25 bool areCollinear(int ind1, int ind2, int ind3)
26 {
27   return (a[ind1].X - a[ind3].X) * (a[ind2].Y - a
       [ind3].Y) ==
28     (a[ind2].X - a[ind3].X) * (a[ind1].Y - a[ind3
         ].Y);
29 }
30
```

```
31 //Are all points collinear?
32 bool allCollinear()
33 {
34   FOR(i, 3, n)
35     if (!areCollinear(1, 2, i)) return false;
36   return true;
37 }
38
39 //Marks points covered by these two lines then
       checks if the remaining are collinear
40 //If yes, returns two points from the remaining
41 bool areCoveredBy(int ind1, int ind2, int ind3,
     int ind4, int &ind5, int &ind6)
42 {
43   vector <bool> covered(n + 1);
44   FOR(i, 1, n)
45     if (areCollinear(ind1, ind2, i) ||
         areCollinear(ind3, ind4, i))
46       covered[i] = true;
47   vector <int> remaining;
48   FOR(i, 1, n)
49     if (!covered[i])
50       remaining.push_back(i);
51   FOR(i, 2, (int) remaining.size() - 1)
52     if (!areCollinear(remaining[0], remaining[1],
          remaining[i]))
```

```
53          return false;
54      if (remaining.size() > 1)
55      {
56        ind5 = remaining[0];
57        ind6 = remaining[1];
58      }
59      else
60        if (remaining.size() == 1)
61        {
62          ind5 = ind1;
63          ind6 = remaining[0];
64        }
65        else
66        {
67          ind5 = ind1;
68          ind6 = ind2;
69        }
70      return true;
71  }
72
73  void writeData(int ind1, int ind2, int ind3, int
        ind4, int ind5, int ind6)
74  {
75      ofstream fout(OUT_FILE);
76      fout << "A torpek meg fognak menekulni." <<
           endl;
```

```
77    fout << ind1 << " " << ind2 << endl;
78    fout << ind3 << " " << ind4 << endl;
79    fout << ind5 << " " << ind6 << endl;
80  }
81
82  void noSolution()
83  {
84    ofstream fout(OUT_FILE);
85    fout << "A torpek veszelyben vannak." << endl;
86  }
87
88  bool checkSolution(int ind1, int ind2, int ind3,
       int ind4)
89  {
90    int remaining1, remaining2;
91    if (areCoveredBy(ind1, ind2, ind3, ind4,
         remaining1, remaining2))
92    {
93      writeData(ind1, ind2, ind3, ind4, remaining1,
           remaining2);
94      return true;
95    }
96    return false;
97  }
98
99  int main()
```

```
100 {
101   readData();
102
103   if (allCollinear())
104   {
105     writeData(1, 2, 1, 2, 1, 2);
106     return 0;
107   }
108
109   //If not all points are collinear, find three
        points which are not collinear
110   int A = 1, B = 2, C = 3;
111   while (areCollinear(A, B, C)) ++C;
112   if (checkSolution(A, B, A, C)) return 0;
113   if (checkSolution(B, A, B, C)) return 0;
114   if (checkSolution(C, A, C, B)) return 0;
115
116   //Find a point outside of any line of the
        triangle
117   int D = 1;
118   while (areCollinear(A, B, D) || areCollinear(B,
        C, D) || areCollinear(A, C, D)) ++D;
119
120   if (checkSolution(A, B, C, D)) return 0;
121   else
122   {
```

```
123       int E = 1;
124       while (areCollinear(A, B, E) || areCollinear(
              C, D, E)) ++E;
125       if (checkSolution(A, B, E, C)) return 0;
126       if (checkSolution(A, B, E, D)) return 0;
127       if (checkSolution(C, D, E, A)) return 0;
128       if (checkSolution(C, D, E, B)) return 0;
129     }
130
131     if (checkSolution(A, C, B, D)) return 0;
132     else
133     {
134       int E = 1;
135       while (areCollinear(A, C, E) || areCollinear(
              B, D, E)) ++E;
136       if (checkSolution(A, C, E, B)) return 0;
137       if (checkSolution(A, C, E, D)) return 0;
138       if (checkSolution(B, D, E, A)) return 0;
139       if (checkSolution(B, D, E, C)) return 0;
140     }
141
142     if (checkSolution(A, D, B, C)) return 0;
143     else
144     {
145       int E = 1;
146       while (areCollinear(A, D, E) || areCollinear(
```

```
            B, C, E)) ++E;
147       if (checkSolution(A, D, E, B)) return 0;
148       if (checkSolution(A, D, E, C)) return 0;
149       if (checkSolution(B, C, E, A)) return 0;
150       if (checkSolution(B, C, E, D)) return 0;
151     }
152
153     noSolution();
154     return 0;
155 }
```

5.5.3. Kovács János

```
 1 #include <vector>
 2 #include <fstream>
 3 #include <queue>
 4
 5 #define FOR(i, a, b) for(int i = (a); i <= (b);
       ++i)
 6 #define X first
 7 #define Y second
 8
 9 #define IN_FILE "kovacsjanos.in"
10 #define OUT_FILE "kovacsjanos.out"
11
12 using namespace std;
13
```

```
14 ifstream fin(IN_FILE);
15 ofstream fout(OUT_FILE);
16 int test, n, m, k, p;
17 vector < vector <int> > g;
18 vector <int> from, to, bitcount(32);
19 vector < vector < vector <int> > > dist;
20
21 void readData()
22 {
23   fin >> n >> m;
24   g.clear(), g.resize(n + 1);
25   FOR(i, 1, m)
26   {
27     int x, y;
28     fin >> x >> y;
29     g[x].push_back(y);
30     g[y].push_back(x);
31   }
32   fin >> k >> p;
33   from.clear(), to.clear();
34   FOR(i, 1, k)
35   {
36     int x, y;
37     fin >> x >> y;
38     if (x != y)
39     {
```

```
40          from.push_back(x);
41          to.push_back(y);
42        }
43      }
44      k = from.size();
45  }
46
47  int bf()
48  {
49      dist.clear();
50      dist.resize(n + 1, vector < vector <int> > (1
            << k, vector <int> (1 << k, -1)));
51      queue < pair <int, pair <int, int> > > fifo;
52      fifo.push(make_pair(1, make_pair(0, 0)));
53      dist[1][0][0] = 0;
54      int allObjects = (1 << k) - 1;
55      while (dist[n][0][allObjects] == -1)
56      {
57          pair <int, pair <int, int> > node = fifo.
                front();
58          int room = node.X;
59          int hand = node.Y.X;
60          int done = node.Y.Y;
61          //Move
62          FOR(i, 0, g[room].size() - 1)
63          {
```

```
64          int next = g[room][i];
65          if (dist[next][hand][done] == -1)
66          {
67            dist[next][hand][done] = dist[room][hand
                ][done] + 1;
68            fifo.push(make_pair(next, make_pair(hand,
                  done)));
69          }
70        }
71        //Pick up
72        if (bitcount[hand] < p)
73          FOR(i, 0, k - 1)
74            if (from[i] == room && ((hand & (1 << i))
                  == 0))
75            {
76              int newHand = hand | (1 << i);
77              if (dist[room][newHand][done] == -1)
78              {
79                dist[room][newHand][done] = dist[room
                    ][hand][done] + 1;
80                fifo.push(make_pair(room, make_pair(
                    newHand, done)));
81              }
82            }
83        //Leave
84        FOR(i, 0, k - 1)
```

```
85          if (to[i] == room && ((hand & (1 << i)) !=
                0) && ((done & (1 << i)) == 0))
86          {
87            int newHand = hand ^ (1 << i);
88            int newDone = done | (1 << i);
89            if (dist[room][newHand][newDone] == -1)
90            {
91              dist[room][newHand][newDone] = dist[
                    room][hand][done] + 1;
92              fifo.push(make_pair(room, make_pair(
                    newHand, newDone)));
93            }
94          }
95        fifo.pop();
96      }
97      return dist[n][0][allObjects];
98  }
99
100 int main()
101 {
102   FOR(i, 1, 31) bitcount[i] = bitcount[i >> 1] +
          (i & 1);
103    fin >> test;
104   FOR(t, 1, test)
105   {
106     readData();
```

```
107         fout << bf() * 13 << endl;
108     }
109     return 0;
110 }
```

Irodalomjegyzék

[1] Adelson-Velskii, George M., and Evgenii Mikhailovich Landis. An algorithm for organization of information. *Doklady Akademii Nauk*, 146, 2, 263–266. Russian Academy of Sciences, 1962.

[2] Bellman, Richard. The theory of dynamic programming. *Bulletin of the American Mathematical Society* 60, 6 (1954): 503–515.

[3] Bellman, Richard. Dynamic programming. *Science* 153, 3731 (1966): 34–37.

[4] Bentley, Jon Louis. Solutions to Klee's rectangle problems. Unpublished manuscript (1977): 282–300.

[5] Galler, Bernard A., and Michael J. Fisher. An improved equivalence algorithm. *Communications of the ACM* 7, 5 (1964): 301-303.

[6] Hierholzer, Carl, and Chr Wiener. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6, 1 (1873): 30–32.

[7] Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest *Algoritmusok*. Műszaki Könyvkiadó, Budapest, 1997.

[8] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Új algoritmusok*. Scolar kiadó, Budapest, 2003.

[9] De La Briandais, Rene. File searching using variable length keys. *Papers presented at the the March 3-5, 1959, western joint computer conference*, 295–298. ACM, 1959.

[10] Dijkstra, Edsger W. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959): 269–271.

[11] Dinitz, Yefim. Dinitz algorithm: The original version and Even's version. *Theoretical computer science*, 218–240. Springer, Berlin, Heidelberg, 2006.

[12] Dirichlet, Peter Gustav Lejeune. Einige neue Sätze über unbestimmte Gleichungen. 1834.

[13] Edmonds, Jack, and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)* 19, 2 (1972): 248–264.

[14] Fenwick, Peter M. A new data structure for cumulative frequency tables. *Software: Practice and Experience* 24, 3 (1994): 327–336.

[15] Floyd, Robert W. Algorithm 97: shortest path. *Communications of the ACM* 5, 6 (1962): 345.

[16] Ford Jr, Lester Randolph, and Delbert Ray Fulkerson. Solving the transportation problem. *Management Science* 3, 1 (1956): 24–32.

[17] Guibas, Leo J., and Robert Sedgewick. A dichromatic framework for balanced trees. *19th Annual Symposium on Foundations of Computer Science* (sfcs 1978), 8–21. IEEE, 1978.

[18] Hopcroft, John E., and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing* 2, 4 (1973): 225–231.

[19] Hopcroft, John E., and Jeffrey D. Ullman. Set merging algorithms. *SIAM Journal on Computing* 2, 4 (1973): 294–303.

[20] Karzanov, Alexander V. An exact estimate of an algorithm for finding a maximum flow, applied to the problem "on representatives". *Problems in Cybernetics* 5 (1973): 66–70.

[21] Kelley Jr, James E., and Morgan R. Walker. Critical-path planning and scheduling. *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, 160–173. ACM, 1959.

[22] Ionescu, Klára. *Bevezetés az algoritmikába.* Kolozsvári Egyetemi Kiadó, 2007.

[23] Ionescu, Klára, and Pătcaş, Csaba. Algoritmusok hatékonyságának növelése a bináris keresés elvének alkalmazásával. *Műszaki szemle*, 43(3):7–14, 2008.

[24] Jarnik, Vojtech. About a certain minimal problem. *Práce Moravské Prirodovedecké Spolecnosti* 6 (1930): 57–63.

[25] Knuth, Donald E., James H. Morris, Jr, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM journal on computing* 6, 2 (1977): 323–350.

[26] Kruskal, Joseph B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* 7, 1 (1956): 48–50.

[27] Lee, Chin Yang. An algorithm for path connections and its applications. *IRE transactions on electronic computers* 3 (1961): 346–365.

[28] Moore, Edward F. The shortest path through a maze. *In Proc. Int. Symp. Switching Theory*, 1959, 285–292. 1959.

[29] Pătcaş, Csaba, and Ionescu, Klára. Algorithmics of the knapsack type tasks. *Teaching Mathematics and Computer Science* INFODIDACT:37–71, 2008.

[30] Prim, Robert Clay. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36, 6 (1957): 1389–1401.

[31] Roy, Bernard. Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences* 249, 2 (1959): 216–218.

[32] Ryabko, B. Ya. A fast on-line adaptive code. *IEEE transactions on information theory* 38, 4 (1992): 1400–1404.

[33] Tarjan, Robert E., and Jan Van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM (JACM)* 31, 2 (1984): 245–281.

[34] Warshall, Stephen. A theorem on boolean matrices. *Journal of the ACM*. 1962.

[35] Williams, John William Joseph. Algorithm 232: heapsort. *Commun. ACM* 7 (1964): 347-348.

[36] *Gazeta de informatică*, Computer press Agora, Tg. Mureş, 1993-2004.

Ezt a feladatgyűjteményt elsősorban segédanyagnak szánjuk az *Informatikafeladatok megoldása haladó módszerekkel* nevű tantárgyhoz, amely opcionálisan választható a Babeș–Bolyai Tudományegyetem Matematika és Informatika Karán elsőéves egyetemisták számára. A tantárgy fő célja a hallgatók megismertetése az algoritmikai jellegű egyéni és csapatos programozói versenyekkel, valamint olyan haladóbb szintű algoritmusok bemutatása, amelyekre más tantárgyak keretein belül nem kerül sor.

Könyvünk hasznos lehet azok számára is, akik érdeklődnek az informatika-versenyek és az algoritmusok világa iránt, és mélyíteni szeretnék tudásukat ezen a téren.

A bemutatott feladatok különböző megyei, megyeközi, országos és nemzetközi szintű versenyekről származnak az elmúlt 15 évből, mindegyiket részletes magyarázat és C/C++ nyelvben implementált megoldás kíséri.

ISBN 978-606-37-0687-5